University of Central Florida

Sponsored By:

Close to Home

Project Documentation

Group 23
April 26th, 2014

Josh Early
Marc Garcia
Nicholas Godfrey
Daniel Krummen

Table of Contents

1.0 Executive Summary Page 1

2.0 Project Description Page 2

2.1 Objectives and Goals Page 3

2.2 Project Specifications Page 4

2.2.1 Software Specifications Page 5

2.2.2 Hardware Specifications Page 5

2.2.3 User Constraints Page 5

2.2.3 Budget And Financing Page 6

2.2.5 Project Milestones Page 7

3.0 Research and Background Page 8

3.1 Smartphone App Page 9

3.1.1 Design and Platform Page 12

3.1.2 App to Web Server Communication Page 18

3.1.3 App to Microcontroller Communication Page 21

3.2 Web Server Page 22

3.3 Hub Page 27

3.3.1 Texas Instruments MSP430 Line Page 33

3.3.2 ATMega328 Page 36

3.3.3 Intel NUC Page 38

3.3.4 PandaBoard Page 42

3.3.5 BeagleBoard/BeagleBone Page 46

3.3.6 Raspberry Pi Page 48

3.4 C2H Modules Page 52

3.4.1 Microcontroller Page 56

3.4.2 In-Wall Modules Page 57

3.4.3 120V Plug Modules Page 59

3.4.4 240V Plug Modules Page 61
3.4.5 Computer Modules Page 61
3.4.6 Lock Modules Page 61
3.4.7 Occupancy Sensor Page 61
3.4.8 Ultrasonic Sensor Page 62
3.4.9 Photoelectric Sensor Page 62
3.4.10 IR Motion Detector Page 63
3.4.11 Sensor-Type Conclusion Page 63
3.6 Circuitry Housing Page 65
3.8 Wireless Communication Dongle Page 66
3.8.1 Bluetooth (IEEE 802.15.1) Page 67
3.8.2 Wi-Fi (IEEE 802.11) Page 68
3.8.3 ZigBee (IEEE 802.15.4) Page 70
4.0 Software Design Details Page 71
4.1 Android Application Design Details Page 72
4.1.1 Concept of Operations Page 74
4.1.2 System Requirements Page 76
4.1.3 High Level Design Page 78
4.1.4 Low Level Design Page 79
4.1.5 Test Plan Page 81
4.2 Possible Webserver Architecture Page 82
4.2.1 Apache Webserver Page 83
4.2.2 MySQL Database Page 84
4.2.3 PHP Page 87
4.2.4 WebIOPI Page 89
4.2.5 Webserver Specifics Page 90
5.0 Hardware Design Details Page 91
5.1 Hub Page 94

5.2 C2H Modules Page 98
5.3 Microcontroller Page 102
5.4 Lock Status and Amp Meter Page 108
5.5 Servo Page 110
5.6 Kill Switch (Relay) Page 115
5.7 Power Supply Page 117
6.0 Prototype Construction and Architecture Page 121
7.0 Prototype Testing Page 123
7.1 Component/System Testing Specifications Page 126
7.2 Who's Who of Testing Page 130
7.3 Testing Procedures Page 131
8.0 Project Operation Page 132
9.0 Project and Group Organization Page 133
10.0 Project Summary and Conslusion Page 134
Appendix A : Copyright Permissions
Appendix B : Datasheets

1.0 Executive Summary

The Close to Home project is, at its core, an effort to provide home users a way to not only monitor their house to ensure its security, but also control several aspects of the house to save on their monthly power bills, and to provide them with peace of mind. The Close to Home system is one that will be able to be integrated seamlessly into a user's home, and provide them with all of the features of the system in a non-invasive and non-obstructive way. Having complete control over your home will be simple and easy with the Close to Home system using the Android Application that accompanies the system. Paired with the central hub somewhere close to the home's internet router, and a few modules installed throughout the home to communicate with, a user will be able to be the master of their own domain.

Our objectives for Close to Home are essentially to provide a system for the home that saves money on a user's power bill every month, while providing them with security and monitoring features at their fingertips. Ultimately, the system will provide more saving benefits than the cost of using the service, as well as providing at least 40% more power savings than upon a housing system that does not have Close to Home installed. We will be providing evidence of this with a scaled-down version of a home and its power uses. We also want to make the software involved very user-friendly, so it is not difficult to set up the Close to Home System to a point where it is providing the user with the most power savings and security.

Through much investigation and deliberation, we decided upon having a Raspberry Pi with a ARM v6 processor as the hub of the Close to Home system, with MSP430 in-wall modules placed throughout various locations in the home, all communicating over the ZigBee wireless interface. The application that will be used to monitor and control this system will be an Android Application that can be easily downloaded and used on any Android smartphone or tablet device. The combination of these three specifications lead to a Close to Home system that was within our expected results, and will provide adequate services to the end user. As well, given that our design is so modular, it would not be difficult to scale up our small diorama to a full sized house and run into little to no issues.

2.0 Project Description

Our group wanted to have a project that would incorporate a large piece of software with at least two different kinds of hardware and an android application. An automated smart house would allow us to implement multiple kinds of hardware (motors, sensors, etc.), design an underlying software to control it all, and implement the whole thing with an android application. We will never have to be individually responsible for one portion of the project, we all know how to incorporate the parts as a whole, and we all have something unique to gain.

We are creating an automated, smart house that will include convenient user control through an Android application. We hope to bring you a better home experience with improved security and energy savings. A central hub will control the smart house, which is still currently in the design process. It will receive commands from both an Android app via a web server, and through an in-house tablet loaded with the app, which can be detached and carried around. The biggest advantage with this design, is being able to fall asleep knowing everything is locked up tight. Each room will be equipped with a break-beam eye system that will monitor the rooms to keep track of which rooms are occupied and which rooms are vacant. From here, user defined settings will determine what, if anything, should be on in the room. This is where we plan to see a majority of our cost savings.

We also know that you are not always at your house, and peace of mind is hard to get when you're away on vacation, or even gone for the day for work, this is where the phone app comes in. The Android app will give you the same control the in-house control panel would give you, including the ability to lock/unlock doors, monitor your appliances, or even program timers. This will deliver convenient home control to anyone with an Android phone. Various levels of control can be set for each user allowing for a more hierarchical control system.

Imagine the day when you go on vacation and the only thing on is your security system, but you can remotely access your door lock to let your house sitter in. Never worry about forgetting to lock the front door or shut off the lights when you leave the room, because the automated house will handle that for you. Never worry about your home being hot when you get home from work, you can turn the A/C on just before you head home. The possibilities are as limitless as the sensors and

controls we can put on your home. Everyone makes a big deal about checking your a/c, remotely logging into their desktop, or keylessly unlocking their doors, but imagine controlling everything with just one app.

2.1 Objectives and Goals

There are five main goals we plan to accomplish with our project:

-Provide energy savings to the user.
-Provide a robust and easy to use security system to the user.
-Provide an automated home experience.
-Bring the convenience of your phone to your home.
-Incorporate everything together with an Android Application.

We plan to incorporate every aspect of the average home into your smart phone with one application. This will not only provide energy savings and ease of mind, but also a convenience which you have only seen in the movies. Security systems, smart air conditioning, and smart phone door locks are just the tip of the iceberg. Our appliances are already smart; we just need a smart way to connect them.

So, we plan to use an underlying smart hub to create a background home automation system. This system will be used to give you piece of mind about day to day activities like remembering to shut the oven off or locking the front door. It will also provide a new level of convenience in the form of turning things on when you are in the room to use them and off when you have not returned in a timely manner. This will provide energy savings in the form of convenience.

The whole system will be tied together with an android application. A large percentage of the US population now uses smart phones and the services that they offer are endless. However, most smart home systems have manifested themselves as one item such as a door lock or a smart a/c. We plan to incorporate the entirety of our home automation system into one android application that can be manipulated with voice commands. Now having everything at your fingertips will finally include controlling your home.

The primary motivations for this project are creating a streamlined home experience while also giving our group a wide exposure to working environments. We could incorporate simple plug and play devices, cut and paste firmware, and utilized premade Android application coding. Instead we are opting to manufacture our own sensors, write our own web server protocols, program our own home automation routine, and incorporate a raspberry pi with every aspect of our project.

2.2 Project Specifications

Date	Description	Author	Comments
11/06/13	Version 1.0	C2H	This reflects a rough draft of our project specifications.
11/15/13	Version 1.1	C2H	A more thorough and expansive document that has double the content of previous rough draft.
11/26/13	Version 2.0	C2H	Nearly all content written. Formal touch-ups and references need correction.
12/01/13	Version 3.0	C2H	Formal document ready for submission in semester 1.
04/27/14	Version 4.0	C2H	Formal Document revised for submission in semester 2.

Table 2.1: Revision History

The following Software Requirements Specification has been accepted and approved by the following:

Printed Name	Title	Date
Joshua Early	Computer Engineer	04/26/14
Daniel Krummen	Computer Engineer	04/26/14
Marc Garcia	Computer Engineer	04/26/14
Bailey Godfrey	Electrical Engineer	04/26/14

Table 2.2: Document Approval

Our home automation project must have the features described here which are fully represented by the Project Requirements Specification. Our project must incorporate hardware sensors/controls into a smart hub which is connected to an Android application via a web server. This will create an automated home system that can be controlled at any moment from the Android application.

The hardware that must be incorporated includes: open/close sensors, locked/unlocked sensors, temperature sensors, trip beam sensors, voltage sensors, a smart hub, a smart thermostat, and on/off controls. The doors and windows require open/close sensors. The locks require locked/unlocked sensors and controls for locking/unlocking them. The lights and fans require on/off sensors and on/off controls. Each doorway must have a tripwire sensor to monitor the presence of people throughout the house. Each room must have a temperature sensor in order to give a more accurate feeling for average, experienced temperature throughout the house. The washer/dryer must have on/off sensors on them. Each major appliance and outlet must have a voltage monitor on it to create an accurate feeling for power consumption. Each kitchen appliance must have an on/off sensor and an on/off control. The outlets in each room must have on/off sensors and on/off controls.

All of the above listed hardware elements will be incorporated into our smart hub in order to allow for an automated, monitored home system. The house will be portioned into rooms based on the boundaries established by the doorway and temperature sensors. The smart hub will keep track of the overall number of people in the house based on the total number people who have entered or exited. It will also keep track of how many people are in each room of the house. Appliances and outlets will be shut off in a room after everyone has left, but no one

has returned in a certain time period. Whenever someone enters a room for the first time, the appliances and outlets will be powered on. The smart hub must also keep track of whether or not appliances are on or off, whether doors or windows are open or closed, and whether or not locks are locked or unlocked. This will allow the user to monitor the security and power consumption of their home. The smart hub must also create an average temperature for the house based on the temperature readings from occupied rooms and use this to control the a/c based on the needs. The smart hub must also notify the Android application whenever the washer or dryer finishes running.

All of the current statuses of the hardware must be relayed via the web server from the smart hub to the Android application. The Android application must also be able to relay commands via the web server back to the smart hub. This allows for anyone with the Android application to monitor the power usage and security of the house while also being able to control any of the hardware. This also opens up the window for voice controlled hardware.

Our project must have security features.

-The doors must be monitored for open/closed.

-The windows must be monitored for open/closed.

-The locks must be automated.

-The status of all security features must be able to be checked.

Our project must have appliance control features.

-The washer and dryer must be monitored for usage.

-The air conditioning must be monitored for efficient timing.

-There must be a way for the a/c to get a more accurate idea of the home's temperature.

-The kitchen appliances must be able to be turned on and off remotely.

-The kitchen appliances status must be able to be checked.

Our project must have home automation features.

-The rooms must be shut off after twenty minutes of no one being in them.

-The rooms must be turned on when a person enters for the first time.

-The locks must be armed when a person exits the home and there is no one else home.

-The locks must be disarmed when the first person connects to the wifi.

-The a/c must be optimized based on home schedule.

Our project must have a smart hub.

- The smart hub must have home automation software on it.
- The smart hub must have a way to connect to the Android application.
- The android application must include controls for all hardware components.
- The android application must incorporate a user id in order to allow multiple users concurrent access.

2.2.1 Software Specifications

Our project must have automated software on the smart hub.

- The automated software must regulate all hardware features autonomously.
- The automated software must keep track of where the residents are in the home.
- The automated software will use this to keep certain portions of the house on or off.
- The automated software will regulate the a/c more efficiently.

Our project must include an android application.

- The android application must be simple to use.
- The android application must have an easy to navigate user interface.
- The android application must include controls for one function per page to reduce clutter.

Our project must incorporate web software.

- The web software will be used to relay communications between the android application and the automated software on the smart hub.

2.2.2 Hardware Specifications

- Our project must have door open/close sensors.
- Our project must have doorway sensors.
- Our project must have smart outlets for appliances.
- Our project must have a control for the A/C.
- Our project must have controls for the lights.
- Our project must have controls for the fans.
- Our project must incorporate voice controls.
- Our project must have automated locking mechanisms.
- Our project must have an android phone.
- Our project must have a smart hub.

2.2.3 User Constraints

- We assume that users know how to use an android phone.
- We assume that users know all of the features of our product.
- We assume that users have our system professionally installed.
- We assume that users have a place of residence.
- We assume that users read the manual for our product.
- We assume that all users have a home network.
- We assume that all users have an android phone.

2.2.4 Budget and Financing

Thanks to the wonderful involvement of the Duke Energy company, we were able to receive funding from them in order to make this project a reality. They have provided us with the request budget below, and we will strive to deliver a product that is up to the standards of such a grand company. Without their support, it would have been unlikely that this project would get off the ground, and we would need to cut many of the features of the Close to Home system.

Part	Cost Per Unit	Qty.	Total
Atmel Microcontrollers	$3.00	50	$150.00
Blank Circuit Boards (30 Piece)	$23.00	2	$56.00
Various Resistors and Capacitors	$20.00	1	$20.00
Raspberry Pi (Hub)	$35.00	1	$35.00
XBee Radio Module	$25.00	50	$1250.00
AA Batteries (20 Pack)	$12.75	2	$25.50
Total:			**$1536.50**

Table 2.1: Initial Budget and Financing for C2H

With the provided funding we plan to start purchasing the parts as soon as the start of the next semester when we hopefully receive funding. In the even that there is a delay in the amount of funding we receive, there is always the chance that we can pay for a module here and there out of pocket, but this shouldn't be entirely necessary. Assuming that the funding will be provided to us directly, the hardest part will be twiddling our thumbs while we wait for the shipments to arrive, rather than actually going about the purchase process.

If, on the other hand, we are to provide a list of parts to Dr. Richie to purchase, that would be adequate as well. We believe it will likely be the latter, and that would actually be preferable, since he will have access to many different resources and locations to purchase from. Given that Dr. Richie has had many years of experience in putting together and bringing Senior Design projects to fruition, he proved to be our go-to question seeker when we were looking for support on any part number or manufacturer question.

2.2.5 Project Milestones

Having a well-thought out and regimented schedule is important for keeping a project on track throughout the entire design and prototyping sections of the project as a whole. It was stressed to us several times by lecturers and previous students of Dr Richie's alike that we will need to focus on rationing our time on working on this project, as the days can slip through our fingertips, and we'll be left with no time to finish what we need to do. What follows is a rough outline of how we plan to divvy up the time left until the final presentation date.

Semester One			
Task	**Start**	**End**	**Duration**
Define Project	08/25/13	09/22/13	29 Days
Research similar projects	09/22/13	09/29/13	7 Days
Draw up specific designs	09/29/13	10/12/13	14 Days
Research how to accomplish them	10/12/13	10/19/13	7 Days
Finalize designs	10/19/13	10/26/13	7 Days
Start putting together parts	10/26/13	11/17/13	21 Days
Section the work into modules	11/17/13	11/24/13	7 Days
Design how to fit the modules together	11/24/13	12/08/13	14 Days
Start the first module	12/08/13	01/05/14	28 Days

Semester Two			
Task	**Start**	**End**	**Duration**
Continue making the modules	01/05/14	01/12/14	7 Days
Test the modules	01/12/14	01/19/14	7 Days
Design the app	01/19/14	02/02/14	14 Days
Make the prototype	02/02/14	02/16/14	14 Days
Make the app work	02/16/14	02/23/14	14 Days
Redesign the system	02/23/14	03/02/14	7 Days
Redesign the app	03/02/14	03/09/14	7 Days
Test the system	03/09/14	03/16/14	7 Days
Fix the system	03/16/14	03/30/14	14 Days
Document how to work the system	03/30/14	04/13/14	14 Days
Demo the system	04/13/14	04/20/14	7 Days

Tables 2.2 and 2.3: Project Milestones by Date and Duration

Following this is a graphical representation of the above milestone chart, with a color coordinated layout in order to present the data as assigned to each specific team member. The below graph can be used to get a rough idea where the biggest sections of work were done, and the largest amount of man-hours were put into the project. Refer to the figure below:

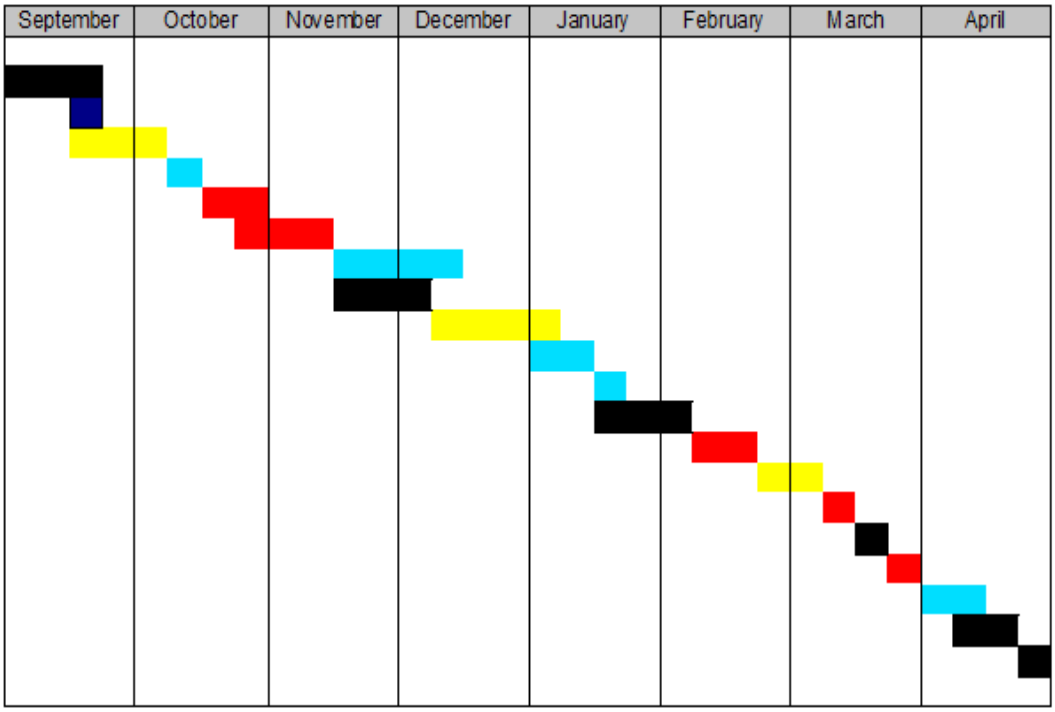


Figure 2.1: Project Milestone Graph

3.0 Research and Background

One of the key aspects of a proper Senior Design project is that half of the project is actually getting to do the hands-on work and actually build a working prototype, and the other half is in the dearth of research that must accompany it. The team spent the majority of the first semester of Senior Design simply looking into all of the available options for the various modules, hubs, programming languages, and platforms upon which to make Close to Home a reality. We will go into them more in depth below.

3.1 Smartphone App

In the design of the application, our main goal is for simplicity and ease of use. The entire purpose for the project is to make the users interaction with their home easy and natural. For this to be possible we had to take into account the tasks our application will be taking care of around the users home. Those tasks include things like turning on or off a light by the flip of a switch. This is already a relatively easy task,

therefore we had to figure out how our application will be more beneficial than doing the task itself. This means that our application must be easy to navigate with as little navigation to perform the desired task. Furthermore, the application must be lightweight in performance. If it takes longer for the application to load than it takes the user to accomplish the task manually, our application becomes less valuable to the user.

3.1.1 Design and Platform

In order for our design to run smooth and natural, we needed to know all the tasks we want our application to perform as well as how much work the user needs to do to complete these tasks using our application. From there we look at how much work the application itself does in the background that the user does not have direct control over.

A list of tasks we would like the users to perform using our application follows:

- Light Status (on/off)
- Door Status (locked/unlocked)
- Display all devices room-by-room along with status

In order for the application to maintain its simplicity and ease of use, we want to only display and allow the user to interact with the above tasks in as few interactions as possible.

Seeing as to how the home is a very personal and secure place, we want to make sure our application is secure especially since the application has an option to not only control the tasks in the users home from the home, but also mobile from anywhere. With this in mind, our application will have a secure log in to the application with various profiles for various users.

Android is our development platform of choice because we believe it give us the ease of use we want. Since android is very widely used world wide, it is a platform users are already used to using. According to Kantar Worldpanel Comtech, [1] Android smartphone sales for 2012-2013 took 51.2% U.S. and 70.1% worldwide This means our application can be widely used and the platform makes the learning

1 http://techland.time.com/2013/04/16/ios-vs-android/

curve for our application an easy one. From a development standpoint, Java is a very versatile language and the Android sdk is very developer friendly. With the Android sdk we have all the resources we need to accomplish our tasks. We are able to customize our user interface to make our application streamlined and fast and easy for the users to use. We have access to built in sensors in the Android-powered devices that we can use in our application to help us measure various environment variables we find useful. Some variables include light levels that we can use to help automate some of our light status controls. We can use the location settings to automate other aspects of the application as well. For connectivity,
Android allows us to connect to the internet via wi-fi and with android 4.0 and higher we can use Wi-Fi Peer-to-Peer which is discussed in section 3.1.3. Android gives us many storage options for our data storage including private internal and public external storage as well as network connection to the web for the mobile aspects of our application.

In our design we have three options for communication between the Android application and the devices themselves. Below we see three possible configurations for getting the users input to the devices themselves.

A)

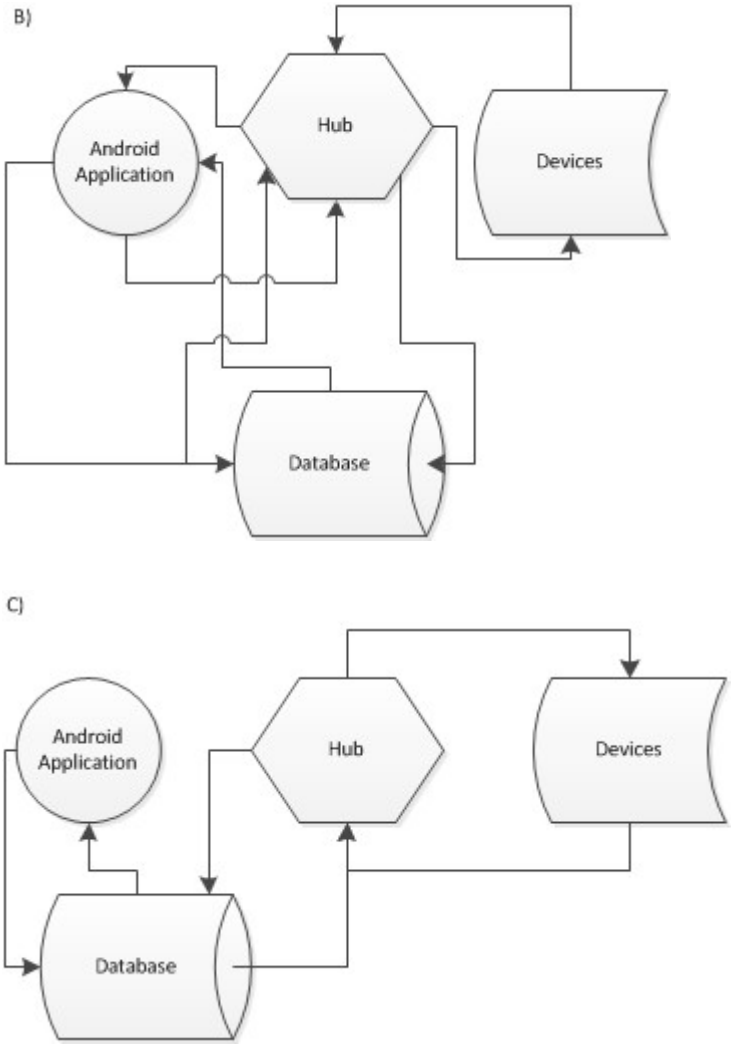


Figure 3.1.1
A) Illustrates a possible configuration for data communication between the application and the devices directly.
B) Illustrates a situation where both the Hub and Database are updated simultaneously for redundancy and portability.
C) Illustrates a configuration where the Android application only interfaces with the database and the database acts as a relay to the Hub.

Above in Figure 3.1.1 we entertain three possible architectures for relaying the data the user inputs into the application to the devices themselves. From here we can examine the pros and cons to such an architecture.

In Figure 3.1.1 A) we see the most simplistic view on how to connect the application to the devices. The idea would be that the device running the application would interface directly to the hub.

The above figures also provide the user with a way to visualize the overall project's flow and what we intend to do with the final project. You can tell from the figures that the system will rely on various parts all working together to make the most coherent experience and develop a device that is able to efficiently and effectively maintain a house's apparatuses throughout the procedure. The database will transmit information to the hub, which will give an ack back to the database when it receives any information from the devices. This can be then relayed via the database to the android application so that the user can tell when changes are made.

Architecture A

Pros	**Cons**
• Simplicity in architectures • Few components to integrated • Data transmission Should be relatively quick	• More computing power is needed from both the Android device and the home device. • More computing power takes more energy thus draining battery • Mobility is questionable at best since there is no web connection we rely on the signal power put out by the Android device and home device. • All data mus be stored on the Android device making portability for other users almost non-existent.

In Figure 3.1.1 B) we see the most complex view on how to connect the application to the devices. The idea would be that the device running the application would interface with both the Hub as well as the Database.

Architecture B

Pros	Cons
• Data is now stored on a database allowing multiple accesses. • Data is redundant so that the application can be used without internet connection if it is lost. • Computing power is now split between more components.	• Redundancy results in longer transmission times. • Data consistency becomes an issue when multiple users are involved in the system. • Programming and coordination become more and more complicated. • More hardware is required to communicate between both the Hub to the Android device and the Database.

In Figure 3.1.1 C) we see the view with most compromise. The idea would be that the device running the application would interface only to the database and relay that data to the Hub where the Hub controls the Devices.

Architecture C

Pros	Cons
• High mobility as mobile devices have web access almost anywhere an average user would be located. • Simple data flow makes for easier programming and less computing power. • Data consistency is no longer an issue with this data flow • Less hardware is required to transmit and receive data.	• No redundancy in data so lack of internet connection hinders the system • Hub must also communicate with both the device as well as the database which may result in longer transmission times.

The android API allows full control over how the application will look to the user. The api allows us to use XML code to design the look and feel

of the application as well as give the user controls of the applications functions that we want to have in out Close To Home system. We created a basic login screen (fig. 3.1.2) using the XML code that takes in the users username and password and parses it as JSON to make the HTTP request and the PHP code verifies that the data matches the users in the database. Once the user is verified, they have access to all the rooms in the household in the room view. This view we made very simple with just room names on a button. This allows easy navigation and little learning curve. All that the user can do is choose a room and the devices should logically follow once a user picks the room. In future development the settings button will allow for more advanced controls such as permissions and scheduling. Android also uses a mechanism called Intents that help make view flow easy and natural for the programmer to use. Once a room is selected, those rooms screens are populated by HTTP GET methods that get all the rooms devices and statuses to display to the user. The user then has control over the widgets and any changes they make here are then posted again back to the database.

Figure 3.1.2
In this figure we see the sample graphical user interface and its screen flow controlled by Andriods Intents.

Android gives us the tools we need to achieve our fluid and natural layout we require through layout tools, input controls and events and menus. Part of the natural easy to use feel has to do with the look and feel of the application as well. There are further tools to customize notifications, graphics, and styles and themes. Being new to the mobile development area we rely heavily on the great support Android provides its developers.[2]

3.1.2 App to Web Server Communication

We want the Close to Home (C2H) application to be secure as well as mobile. For these reasons we will need a way to communicate with the C2H module from anywhere we have web connection on our Android devices. Android makes this easy for us since it provides full support for SQLite databases. With a database, we are able to store user profiles where users data can be encrypted and stored securely yet still be accessible anywhere they have their web connected device. This is how we will manage user permissions the head of the household can set. Once the user is signed into the application, the data the user inputs to the application is stored in the database. From there the C2H hub module will then receive the data via query() (see section 4.2). For security reasons it will be necessary to have internet access for the application to work. In order to ensure the users data sent from the app to the web server is secure, a 256-bit AES encryption is used. SQLCipher is an open source library that is compact in size and has low overhead. It also supports Android OS 2.x - 4.1 and is therefore fitting to our simplified yet secure objective.

Our application needed to to basic create, read, update, and delete operations on the database. We are able to do this using PHP scripts to connect to the MySQL database. The PHP scripts and WAMP server are all described in section 3.2 because in this section we only highlight the android development aspects that go along with connecting to the database.

From the Android's point of view, it must just have the ability to send and receive information to and from the database. In our case, HTTP works best for us. Hypertext Transfer Protocol (HTTP) was designed to enable the exact type of communications we were looking for between clients (our application) and servers. The two HTTP request methods

2 http://developer.android.com/guide/topics/ui/overview.html

we used are the GET and POST methods that request data from a server and get a response. The GET method requests data from a specified source and the POST submits data to be processed to a specified source.

While it was difficult to decide on exactly what data we wanted to transmit between the client and server, we found that JSON was the best fit for our application. JSON or JavaScript Object Notation, used human-readable text to transmit the data between server an application. JSON is a language-independent data format which made it easily applicable to our application. The JSON notation was not only important for data transfer but it was also used to generate the applications user interface since we want the application to display the modules and their status for that particular household. This JSON format data is parsed in a special parsing class in the android JAVA code using the HTTP GET and POST protocols. The application only needs to see this data that is given and will only send the data in the same format. It is the PHP codes that will handshake with the database and perform all the database manipulations.

3.1.3 App to Microcontroller Communication

There are a few possibilities for this communication to be possible. There is the possibility of using bluetooth or nfc but there are the issues of power consumption and communication distance. Another possibility is using a server as the only means of communication between the application and the controller. While this option is one way we plan on using the application to communicate with the controller from anywhere there is a mobile network connection, we also want a secure local connection for when the user is at home. For this there is an API that is available in Java called java.net. With that we could set up a serversocket that can communicate via wi-fi with the wi-fi enabled arduino. In order to have the arduino connected to the wi-fi network we could obtain a WifiShiled that connects the microcontroller to the local wi-fi thus, enabling app to microcontroller communication via wi-fi.

3.2 Web Server

There are multiple options which can be utilized to provide over the web communications from our smart hub to our android application. The

option that we decide on will depend on the amount of features needed by our communications. In our research for web server functionality, we have made the assumption that we will utilize a Raspberry Pi as our smart hub. Three of the most likely candidates are listed here and discussed in further details below:

WebIOPI

"A fully integrated Internet of Things framework for the Raspberry Pi" This is an open source communications framework that can be utilized to send data between an Android application and scripts currently in execution on the Raspberry Pi. This will be the simplest to utilize, but will limit how we create our algorithms for our smart hub. This incorporates its own user authentication, javascript and python libraries, and supports multiple hardware devices and protocols for communicating with them. More information can be found here: [3]

Custom Firmware

If we need the robustness of full blown java, WebIOPI will not be an option since it only has script libraries. However, we can utilize java to create classes for external communications and security. This can be done by utilizing Java Web Services by embedding Java SE on the Raspberry Pi in order to establish a connection with your Java Web Services codes. A simple run down of Java Web Services can be seen here: [4]
Implementing Java Web Services through the Raspberry Pi can be seen here: [5]

PHP Server

The most straightforward way for providing client-server communications is to utilize a PHP Server. This will require us to rent or borrow server space from someone in order to execute a set of PHP scripts. Pre-made or custom PHP scripts can be utilized depending on the requirements of our web communications. The PHP server will utilize PHP scripts in a similar fashion as WebIOPI, except that PHP servers run on the internet whereas WebIOPI runs locally on the Raspberry Pi. This will also require some kind of HTTP markup language such as Apache in order to process the internet requests for the PHP

3 https://code.google.com/p/webiopi/wiki/README
4 http://www.oracle.com/technetwork/java/index-jsp-137004.html
5 http://devilqube.blogspot.com/2013/02/controlling-berry-clipraspberry-pi-from.html

server. A simplification of the process can be referenced here at the following URL shown below:[6]
And a tutorial for establishing a PHP server communication can be referenced here: [7]

3.3 Hub

The Close to Home system is, at its core, a network of multiple separate units in a star-based topology. At the center of this network is the centralized hub, which directs and monitors all of the household devices in the system. Each individual device does not perform very many computations on their own. For the most part, the various door, light, and lock sensors merely report periodically to the central hub with a few bits of information so that the hub can know what commands to give back to them next.

Considering the workload that the central hub would be required to do, the group had to make a tough choice as to what piece of hardware should be used. We needed a device that had the computation power to handle the incidents where multiple units on the network would report for their next command, and the ability to run a set of code that routinely checks for room and lock status throughout the house. As well, the hub had to be low in power usage, and have the ability to sleep between cycles, in order to reduce power consumption. One of the main focuses of the entire system is to incorporate Close to Home into a currently existing home power system and yield a monthly financial increase with the gains that the system provides.

For many of these options, our decision was also affected by the complexity of the developmental environment. For example, we could have simply used a Laptop PC as the central hub of the project. This would effectively negate the cost—assuming we could borrow the personal laptop of a member of the group—and thus provide us with a powerful platform to monitor and maintain the house's various controls with. An ultimate goal of this senior design project, however, was to create a product that would hone our skills as developers and engineers, and give us something to put on our resumes when we

6 http://www.xda-developers.com/android/intro-guide-to-android-client-server-communication/
7 http://www.stanford.edu/class/ee368/Android/Tutorial-3-Server-Client-Communication-for-Android.pdf

finally had the completed project. Thus, we always looked at each “hub candidate” for the project with the mindset of 'Will this be something I would like to brag about later on?' as a driving force. It is relatively simple to go out and purchase an already-built home automation system and monitoring devices, but building them from the ground up from scratch is what truly is an achievement.

Another aspect that we wanted to consider was whether or not the developmental board had enough support in online reference material and online forums. If the group decided to go with a board that was more obscure in the market, or only had support from the vendor, we would be faced with a problem in the event of running into a troubleshooting issue that was specific to the device. If instead we opted for a more widespread board that has much more online support and any kind of following on online discussions, we could find support from others that share the passion for the device, or get advice from others who did similar projects and what kind of obstacles they ran into. We wanted a board that we could stand on the shoulders of giants with, and not one that we would have to dig too deep for information on.

3.3.1 Texas Instruments MSP430 Line

One of the boards that we considered initially was the Texas Instruments (TI) line of developmental boards, since TI is a trusted name in lower-level development and computing. TI has a various collection of wireless developmental boards in their eZ3430 product catalog, and using anything from the MSP430 product line would be desirable for our team members, due to the experience that they had with the board during the Computer Architecture class offered as part of the UCF curriculum. Taking one of the prime candidates as an example —the eZ340-RF2500 seemed a great choice due to the low price point of $49.00 for the unit, and the slew of features that made it seem ideal for the Close to Home system.

However, the RF2500 was very limited in its scalability. The board had very few expansion slots to incorporate more features than originally planned by Texas Instruments. And what expansion slots there were only had compatibilities with TI-related products—something that would severely inhibit the developmental stage of our design process.

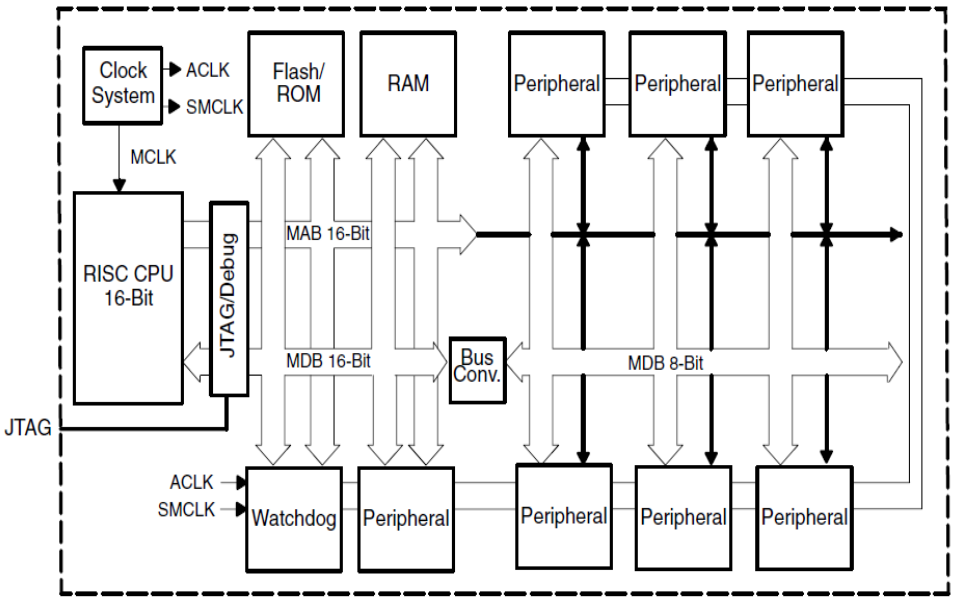


Figure 3.1: MSP430 Architectural Layout Printed with permission from cnx.org

3.3.2 ATMega328

An ATMega328 processor-based microboard was a platform that the group considered as a hub for some time, and for good reason. It is a highly supported developmental board and there are hundreds, if not thousands of home projects that run on the Arduino Uno version of this board. However, since we wanted to have a project with the potential to be scaled up to a full-sized home, we needed a board with a little more processing power. The ATMega328 processor within the Arduino Uno only has roughly 16Mhz of processing power, and a short 3KB of RAM. Since the Close to Home system would be monitoring every aspect of the house, the Arduino Uno would likely struggle with any kind of scheduling and computation being done on more than one part of the house at a time.

Another reason for leaning away from the Arduino Uno was due to the very limited programming environment that it provides. For the most part, you are restricted to the Uno's singular coding space—a variation of C++. Since we wanted to develop an Android app that worked in tandem with the hub's code, the group needed to be in full control of the developmental environment, providing us with many options for programming languages and platforms.

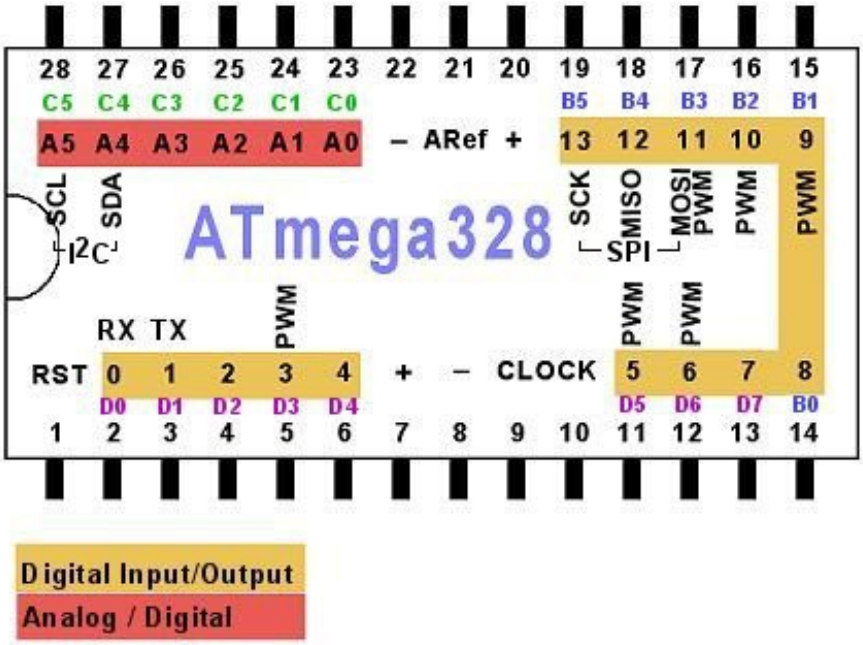


Figure 3.2: ATMega328 Pinout Printed with permission from hobbytronics.co.uk

As you can see from the above image, the ATMega328 hosts a slew of many input, output, clock, SCK, MISO, PWN, and RX/TX pins, which would make it ideal for the application that we were planning on. However, the Arduino Uno that the piece itself sits upon is a bit of a misnomer in terms of the things that it can do and what it cannot do. The Arduino Uno is a perfect computer for a smaller operation, such as managing a bank of servos or attempting to turn a light on and off, but when it came to the need to manage an entire household and make it easy to control, the Arduino Uno fell short.

As well, it was difficult to choose the Arduino Uno again due to the programming language restrictions involved. Without the ability to branch out into a language that suited our needs, like Python, we found it very limiting to only be able to code in the instruction set of C++. Though it is a very powerful language for other applications, we found that it was necessary to have a more diverse language and the majority of which being the fact that you can have close access to lower level controls as well as upper level management wherein the Arduino Uno was the least contender on the list. If we were to start this project over again with a different aim, perhaps the Arduino would have been the proper fit for our needs, but for now it will take a back seat to the other options we have available. As we will invariably outline below in the coming sections.

3.3.3 Intel NUC

On the opposite end of the computing power spectrum, we also considered much higher-end products for use as the Close to Home hub. The Intel NUC (Next Unit of Computing) unit is a small 4”x4”x2” computing device that has the power of an Intel Core i3 Processor. As well, it has a lower power consumption input than a normal laptop or desktop computer of the same processing power, so it looks like a win-win from that perspective. One of the the largest drawbacks of the Intel NUC is the high price point: A steep $290.00 for one of these powerful units. The majority of the budget for Close to Home is in the individual monitoring units that are to be placed around the house. If we decided to stack on top of that a highly expensive Hub device, we would likely run over budget.

Not only that, but keeping the price point low is what is desirable from a consumer prospective as well. If we wanted to get this device into every home nationwide, we would need to make it affordable enough for the average user to purchase. This ultimately leads to the need for a centralized hub that is the bare minimum as far as processing power is concerned, and one that is economically sound.

3.3.4 PandaBoard

One step down in the price point, but with a larger amount of computing power sat the PandaBoard. The PandaBoard is a highly scalable developmental board with a large backing of hobbyists. Finding support for the PandaBoard and scaling it up to the task at hand would be a non-issue. As well, the 1 Ghz processor that this board boasts would have computing power in spades, possibly a large amount of overkill for the project we are looking to do. The PandaBoard comes with on-board Wi-Fi and Bluetooth support, which would turn out to be a blessing and a curse. It's unlikely that we would make full use of Wi-Fi support for the device if we had went with a Bluetooth communication topology for our network, and vice-versa if we had opted to use Wi-Fi as our main form of speaking between devices.

Included with the PandaBoard were also a whole slew of other features: DVI and HDMI Out, 3.5mm audio in and out, a 340Mhz PowerVR SGX540 GPU, and a full 1GB of DDR2 SDRAM. All of these powerful

features and optional hardware add up, however, to bring the PandaBoard to a rather steep price point of $180.00 for the most basic of boards offered. Considering that we would make full use of the various features of this board, and the higher price point in comparison to others we were looking at, the PandaBoard was not the developmental board for us.

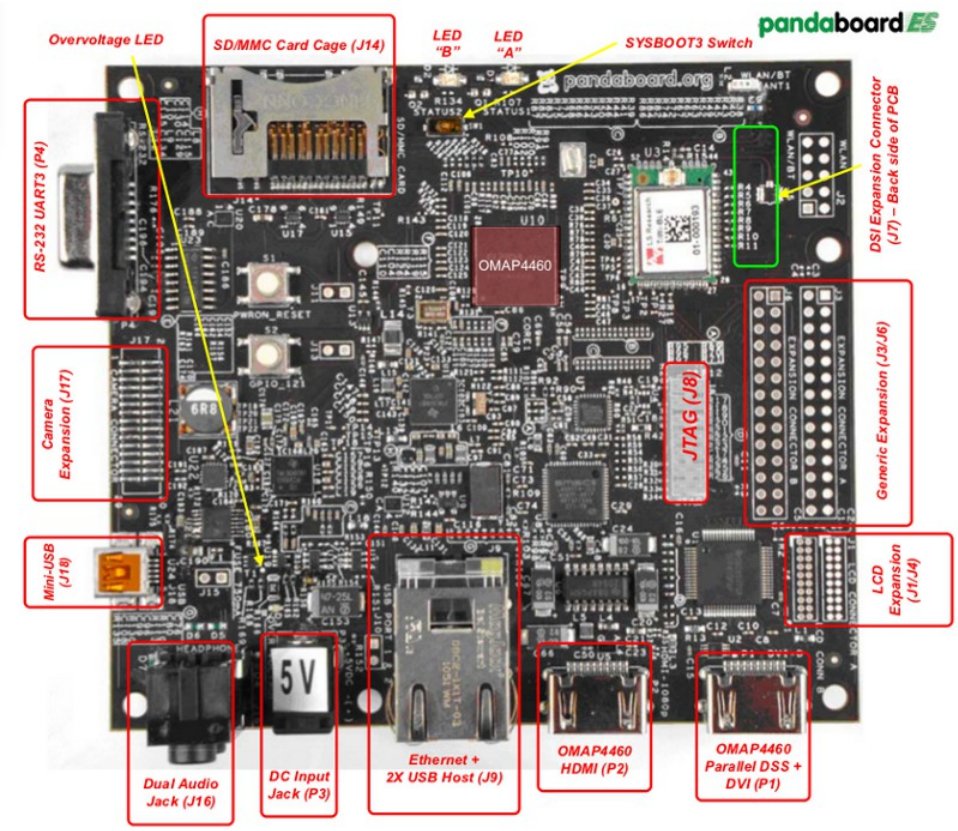


Figure 3.3: PandaBoard Diagram Printed with permission from pandaboard.org

3.3.5 BeagleBoard/BeagleBone

Another runner-up in the board choice was the BeagleBone Black Development Kit. This board had very attractive aspects, such as the lower power consumption compared to other devices, and the large array of analog I/O connections that border the board on the left and right sides. It also had the ability to connect to the internet and upload information—something that later came into play when we made Close to Home accessible while away from the house over the Android app.

However, the biggest downfall of the BeagleBone is the higher price point. At roughly $89.00 for the central unit, it really didn't seem like a worthwhile investment to spend that larger amount of money and not gain a significant benefit in processing power or energy savings. As well, our particular project would make large use of wireless transmission, so the multiple I/O ports that line the side of the board would have gone unused in our application. The BeagleBone (and its brother the larger BeagleBoard) were a very strong candidate, however, and we could easily see the use of one of these boards in a similar project.

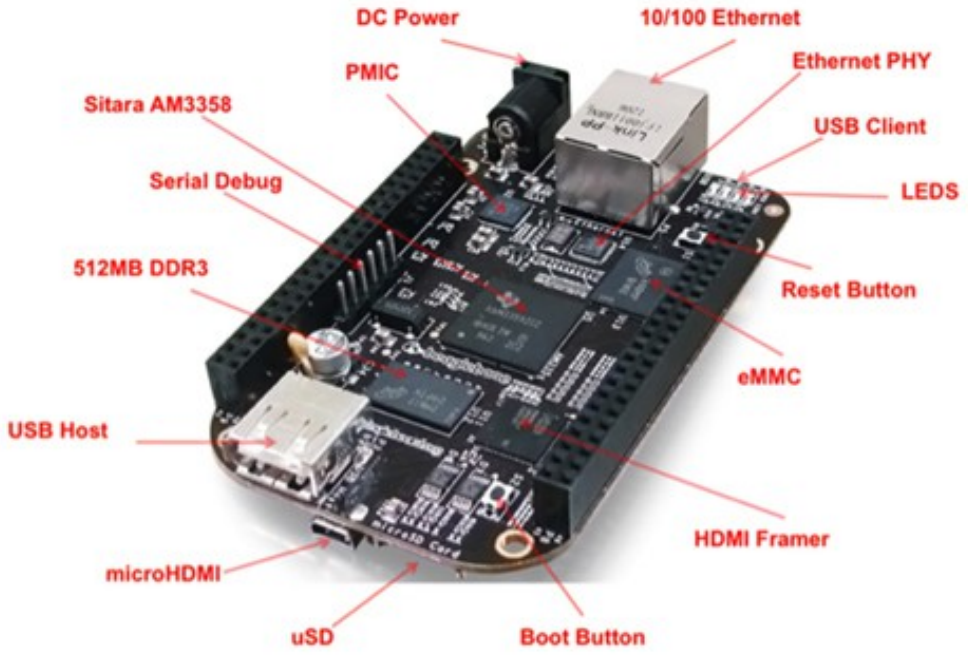


Figure 3.4: BeagleBone Diagram Printed with permission from digikey.com

3.3.6 Raspberry Pi

Taking in all of the downfalls of the other devices, we come to a strict set of requirements for the desired device. We needed a device with a small price point, in order to possibly offer a package that was affordable enough for a consumer to care. We also needed a device with a moderate to low amount of power consumption to keep the

power savings to a beneficial level. Third, we required a device with a modular design so that we could upscale it to a full-sized house if the project ever reached that point. This led us to choose the Raspberry Pi Model B board for our hub unit.

The Raspberry Pi is essentially a small, affordable linux computer that has a surprising amount of processing power in such a tiny package. The Raspberry Pi makes use of the 700Mhz ARM1176JZFS processor, made by Broadcom, which provides the computation power necessary to run smaller builds of Linux, such as Fedora Core and Debian. This is paired with a low-profile set of 512MB of RAM, and a FPU for any floating point calculations that need to be done. This barebones computing environment provided the group with a stable platform to develop our hub application on top of, and allowed us the ability to scale the operation as needed. The Raspberry Pi also runs at a much lower price point compared to the competitors: a mere $35.00 for the board.

Normally, the purchase of the Raspberry Pi would also require the user to purchase an SD Card for permanent storage and a separate power supply, but we were able to make use of an SD Card that a group member had lying around, and a simple Motorola phone charger that someone had at home was perfect for the power supply necessary. Since we could likely assume that the hub would be located next to the home router, using a power supply as opposed to a battery configuration was ideal.

In the end, the Raspberry Pi turned out to be the perfect board for our needs. It provided the group with the necessary environment and scalability that was required to make the ideal system to maintain and monitor the Close to Home system that we made. As well, its modular design let us be more open with our design choices, and didn't lock us in to a specific subset of hardware from the onset. Multiple times during the research and design stages of the project, our group changed our minds about various aspects of the home system—from the form of wireless communication that we planned to use, to the languages that we wanted the program to be written in so that it could communicate with the Android application.

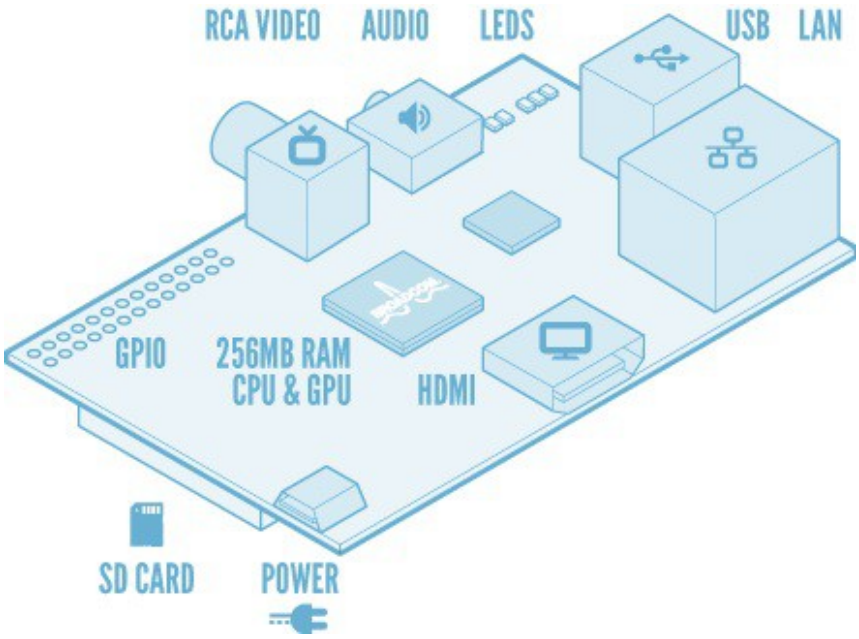


Figure 3.5 Raspberry Pi Diagram Printed with permission from eLinux.org

In Summary, choosing the Raspberry Pi as our developmental board of choice was not a simple one to make. After considering all of the options available, and going over the pros and cons of each board, only then did we reach a final conclusion that would really feature the choice that was proper for the occasion. You can find a small table of the pros and cons of each board below:

	Pros	Cons
MSP430 Line	Cost-Effective. Previous Experience.	Not Widely Supported. Few Addon Modules. Little Wireless Support.
ATMega28	Barebones System. Light Power Use.	Lacking in RAM/CPU. Restricted Coding.
Intel NUC	Very Powerful. Easy to Use.	Overkill for Project. Lack of Difficulty.
Raspberry Pi	Appropriate Power. Low Power Usage. Great Addon Support.	Some Extra Features That Were Not Really Needed.

Table 5.1: Pros vs Cons of Developmental Boards

After the Raspberry Pi was chosen as the appropriate central Hub computing device, it was necessary to outfit it with an XBee unit in order to allow it the ability to transmit information between the modules and the itself effectively. To accomplish this, it was necessary to attach the TX and RX pins on the Raspberry Pi's GPIO pins to an XBee radio module on the appropriate pins, as well as provide the unit with power and ground. While we could have used a simple wire to wire connection and soldered the board's GPIO pins to the XBees' pins, this was less than an elegant solution. Accordingly, there existed a device known as the Slice of Pi that would work as a way to conveniently and easily connect the XBee radio module to the Pi's board. You can see an example of the connected items below:

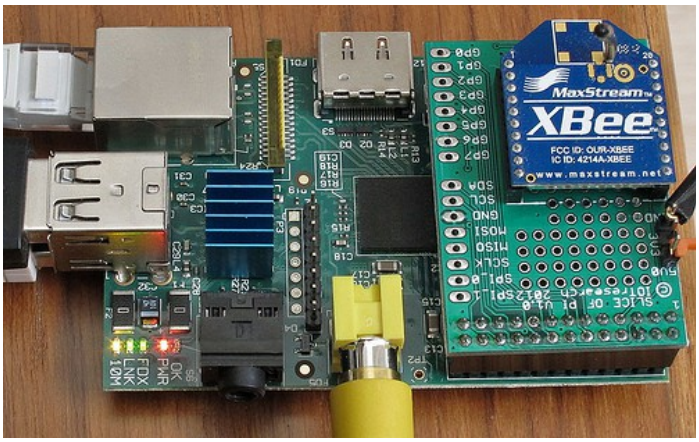


Figure 3.5 Raspberry Pi with attached Slice of Pi and XBee chip

The Slice of Pi proved invaluable, allowing us to troubleshoot a situation where there could have been error in the hub, by eliminating the chance of faulty wiring or loose pins altogether. Once the input brick was soldered into the Slice of Pi and we had proper testing on the final product, it proved quite simple to test the operation of the units using the desktop program X-CTU to send the device serial communications as a test. As well, the X-CTU software could easily interpret and display serial information sent by the Radio Module onboard the Slice of Pi, by using simple Python commands equipped with the PySerial library. This library provided a list of effective and powerful commands that could be used to communicate using serial UART communication over the Xbee's transmission signals, and let us manage the various modules throughout the Close to Home system.

3.4 C2H Modules

For this project, we are aiming to have the entire house outfitted with C2H Modules that will monitor the current status (on/off, lock/unlock) of every appliance, light source and lock of the user's house. We began thinking about a modular adapter that would fit onto every source of energy drain. The original idea was for standardization, that is, a particular design replicated as many times as we need it. Upon further investigation and design, we have decided that we will create five unique modules for each type of power drain.

The first type we will be creating is for in wall lighting, which will be connected to the switches in the wall that control overhead lighting or fans. The next type will be for power drains that plug into a standard wall outlet, lamps and fans in particular. The third type is for computers. We want a special type of module for computers, because we want to be able to send a signal for the computer to safely shut down, instead of just killing the power to it, which can obviously result in data corruption. The next type we want to design, is for appliances that plug into 240v receptacles, these are things like ovens, refrigerators and washing machines. Finally, the last type will be the lock module we will use for the doors. The following sections will break the modules down individually, as one of the advantages of designing individual modules, is we can use different components for each module, meaning we can save money and keep parts more efficient as they are only doing the job they were designed for.

3.4.1 Microcontroller

Each module will of course need a microcontroller in it to control signals, and broadcast information wirelessly. We will have a brief description of what we are looking for in a MCU for each module. Fortunately, since the difference between each module's micro processing needs is minimal, we can use the same microcontroller for each module.

We began our research using what we knew as far as microcontrollers are concerned. The microcontrollers we began research were the MSP430 from Texas Instruments, the AtMega 328 from Atmel, and the PIC32 from MicroChip.

3.4.1.1 PIC32

The PIC32 is a powerful microcontroller made by MicroChip. Something that drew me to the microcontroller in the first place is the amount of RAM and I/O Pins available. After some research I was able to find a model number that we will be using to compare against the other options in microcontrollers. Below is the table with the main values we are focusing on for our project:

PIC32	
RAM Retention	2 uA
Idle	40 uA
Running	250 uA
Flash	16KB
RAM	4KB
Voltage	2.2 to 3.6V
I/O	53

As you can see from the table, we have a lot of memory and a lot of I/O ports to work with. The voltage rating is about average, but slightly above the other two boards we are looking at. Another issue we have picking this board is the amount of current draw while the board is in its "Idle State". These microcontrollers will be spending a good amount of time in an idle state as no user will be switching devices on and off that

often. Furthermore, the PIC32's RAM and I/O are great, however we are hoping to minimize power usage across all of our boards. We can do this by cutting what we need down to a minimum. As the voltage requirement and the RAM retention rate are both high, we will look at other options. Ultimately, this board feels like it would be excessive for our project.

3.4.1.2 AtMega328

The AtMega 328 is the chip that is used in the popular Arduino development board. The Arduino community is very large, and there is a lot of support for use of this chip. The stats for the ATMega328 are impressive; specifically it's RAM Retention rates and amount of Flash Memory. The following a table of the parametric data we are interested in:

AtMega	
RAM Retention	0.5 uA
Standby	90 uA
Running	250 uA
Flash	32KB
RAM	2KB
Voltage	1.8V to 5.5V
I/O	23

Table 3.1: AtMega Specifications

The AtMega328 is another viable microcontroller for our project. We are specifically looking for low power options as we are not looking to drain a bunch of power from the device that is powering it. The obvious problem with this is using a battery is decreased battery life. If the unit is drawing power from the wall, high power requirements could actually result in raising the user's power bill instead of lowering like we are hoping for it to. One of the biggest advantages to using this microcontroller, and the reason we chose is for research, is the large support community behind it. The Arduino is a popular development board and as such, there are is a large amount of documentation, tutorials, and similar projects we can look at to help us complete this project.

3.4.1.3 MSP430

The MSP430 from Texas Instruments is a strong contender for the micro controller for our project. The MSP430 is a very large family, boasting almost one hundred different models. This allows us to pick the exact specifications we need without being excessive, resulting in unnecessary current draw. We are specifically looking at the MSP430F2121. The following is the parametric data we are interested in for this board:

MSP430	
RAM Retention	1 uA
Standby	3 uA
Running	220 uA
Flash	4KB
RAM	256KB
Voltage	1.8V to 3.6V
I/O	16

After comparing the tables on the chips, we see that the standby mode is an option on every chip, however the standby power draw is 90 uA on the AtMega328 which is much higher than the standby of the MSP430 chip, which is 3 uA on the MSP430F2121. We can also see from the tables, that there are large differences in the I/O count as well as the memory for each chip. The Flash memory is a lot larger on the AtMega, but we're looking for more space on the RAM, so based on the table, the MSP430 is the better choice here as well.

For each board we create, we want to create a backup power supply in case of a power outage. It is important to keep a battery on the board, so that the RAM does not reset. Here, we will be choosing an industry standard, the CR2302 3v Lithium Watch Cell. We are specifically looking at the Energizer CR2302 as it has one of the highest mAh capacities of the brands researched. At 240 mAh, rough calculations say that this battery can run the board at the RAM Retention current of 1 uA for about 191 years. This is obviously much higher than what the battery will last, as we may use it for other applications in the board, and battery discharge will dramatically shorten the life of the battery.

After the comparing the chips, and setting low power consumption as our top priority, we are choosing the MSP430. As mentioned before, the MSP430 is a large family. Each model number begins with MSP430 and is then followed by the model number. We started our research with the lowest family and worked our way up until we had every feature we wanted, without having too much excess. We found that the MSP430F1xx family does not have the standby mode we need, and that the MSP430F3xx family had everything we need, plus things we did not, like LCD Controllers. This leaves us with the MSP430G2xx and MSP430F2xx family. These chips have a standby mode that can run at just .3 uA, which is a third of what the AtMegas can run at. Unfortunately, something we need to make some of our modules work is not present. We need some way to measure current to decide whether certain appliances are on or not. The easiest way to do this is to measure the voltage drop across a resistor, which we can do with an ADC. An on-chip ADC will be another feature we will need for our selected chips.

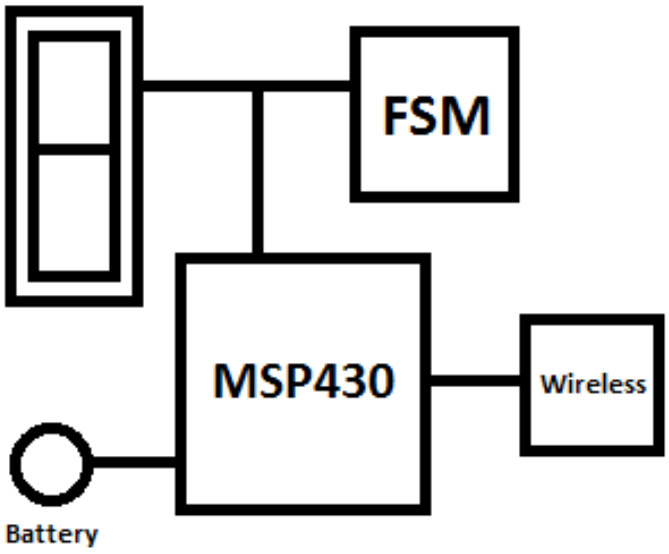
We also need to look at the available I/O ports. Each module will be the same in this aspect. We will have an in/out for the wireless communication, an input for whatever type of status detection system we are using, whether it is to detect the lock/unlock status, or the on/off status of appliances, and an output for a kill switch or servo to change the status of the device to the user's desired status. We also want to put a small watch battery onto each board, to support RAM Retention, in case of a power failure for in-wall devices, and to reduce power drain from the main batteries of the devices actually using battery power. Each family of MSP430 has a very low RAM Retention and Standby, and will be using CR2302 Lithium-Ion battery as a backup power supply to make sure RAM settings are maintained, and of course to keep devices in standby in case of power outages or the like.

3.4.2 In-Wall Modules

For the in-wall module, we would essentially be redesigning the light switch itself. One of the advantages to installing modules into light switches is we can use in wall power to power the modules, so we don't need to worry about an external power source for the module.

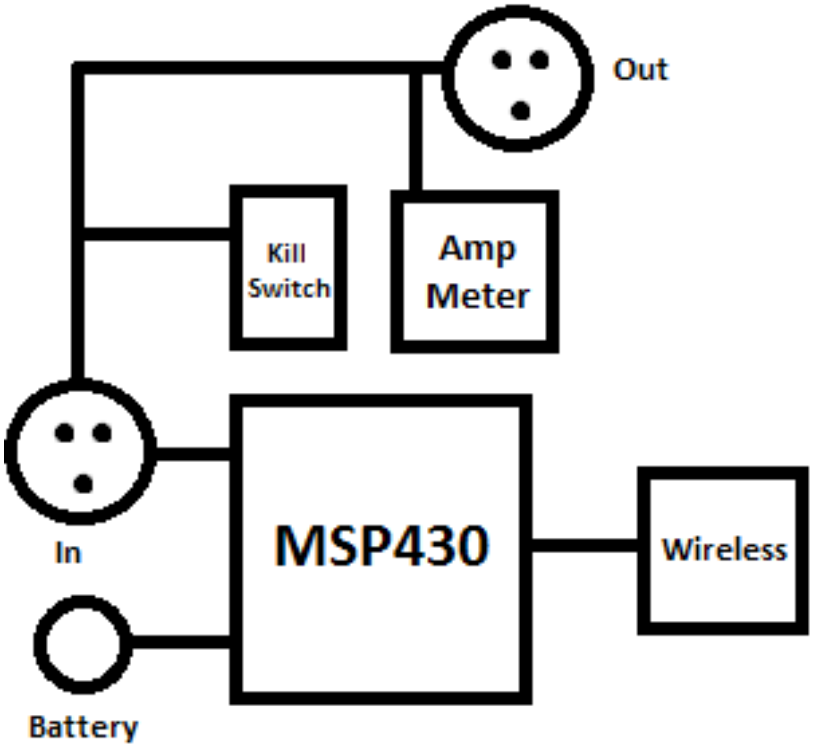
Something that we do need to worry about for modules that use in-wall power, is power drain the module itself. We are looking to reduce the

user's overall power bill by insuring that certain devices are turned off when the user is not home. Therefore, the power draw of the board will need to be less than the power draw from leaving light on for a couple of hours. Preliminary research shows that the average light bulb is the 60 watt bulb. For a bulb like this to be powered from a 120v source, the current drawn would be about .5 amps. We estimate that on average, since some light switches control one light, and others more, that a switch will turn on two lights, meaning that the power draw for powering these two lights would be about 1 amp on average. Initial research shows that the current draw on these boards is far below this, generally in the micro amp range. This means that a board running in standby mode would take almost 40 days before it drew the same amount that the light switch would draw in 1 second. This estimate is a little high of course, because this is just in standby mode. Standby mode on these boards is when the board "sleeps" after a designated period of time. During the operations of the board, the power draw is higher than this for a period of a couple of seconds. Even during this period of increased activity, the current through the board is in the micro amp range, and it is only for a couple of second at the most whenever the micro-controller is pinged by the hub for a status update, or whenever the use manually triggers a status change, such as turning on the switch. For this type of module, since we need a standby mode, but do not necessarily need current measurement, as we can just keep track of the status of the switch via manual or electric toggles, we will be choosing the MSP430F2121. This chip does not have any bells or whistles that we do not need, and has double the Flash Memory of the MSP430F2120. The idea of the components required, and connected to the micro-controller is as follows:



3.4.3 120V Plug Modules

The next type of module we will be designing will be module used for devices like lamps and desk fans. The idea here is that the device will plug into our module, and our module into the wall. Much like the previous module, the monitoring device here will simply be an amp meter. Unfortunately, as there is no real functionality built into the chip to measure a current drawn by the device, as the full current will never run through the chip. What we can do however, is build a circuit outside of the chip, and measure the voltage across a resistor of a known value. We do not know whether the exact value of the on current of the user's device, so we will have to program in a range of currents. The device should essentially draw little or no current while it is off. If the module detects an increase in current, the module will register the device as on. Turning off the device plugged into the module is as easy as killing the power to the device. We will be using a relay to cut the power to the module, thus killing the power to the device attached to the module. A graphical representation of the components is as follows:



While looking around for the proper relay, we have to keep a few thing in mind. First and foremost is that we are working with 120V. While this is a standard value for houses, it is still a large value, so a majority of relays would burnout. We also need to be able to trigger the relay with the power we have. As we are pulling voltage from the wall, it is an easy process to drop the current to any current we need, as long as the MSP430 can process the current without burning out itself. Unfortunately, the current it would take to trigger 120V relays is higher than the 6 mA max current the MSP430F2121 can handle. To remedy this, we will use a transistor to the relay to use the power pulled from the wall to properly trigger the relay. With all of the requisites in mind, we began searching for similar projects online

3.4.4 240V Plug Modules

The fourth type of module will be the module we use for high power appliances like your stove. The only real difference between this module and the module that you use for lamps will be the calculations that must be derived to measure the current, as the voltage from these

receptacles will be double the voltage. There are obvious issues with just shutting off the power to any device plugged into a 240V Receptacle. For example, if we are talking about an oven, loss of power will generally cause the oven clock to fall out of sync with the proper time. We do not view this as a serious problem with the system, as a user who would need to remotely turn their oven on, would generally only do this in an emergency situation. Furthermore, we know that the overall power usage of a stove in idle state is actually much more than what we need to power our devices, so we don't need to do any comparisons about any extra power usage due to our module.

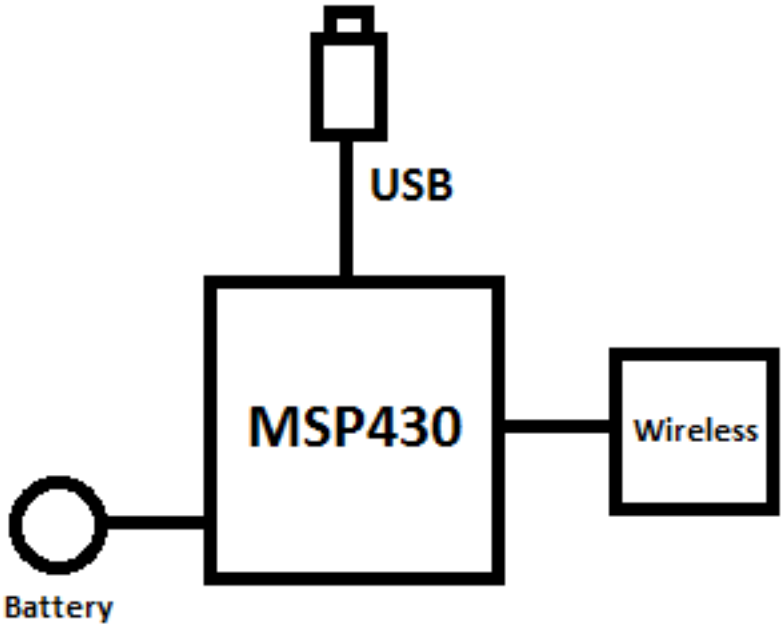
3.4.5 Computer Modules

The third module will be the module will be the module we use to safely shut down computers on command. We are specifically interested in making sure that you are able to turn the computers off as you please, as a computer can use a surprising amount of energy while it is turned on, even when idle and even more surprisingly, when it is asleep. Table 3.2 gives us values on the energy usage of computers.

Mode	Usage
Active	115 Watts per Hour
Idle	70 Watts per Hour
Sleep	5 Watts per Hour

Table 3.2: Printed with permission from Progress Energy Inc.

As you can see, the cost of a computer can weigh heavily on your power bill. We are still doing research on this, however we are aware that every computer has a command that can be enacted from a peripheral. This module will be plugged into a computer via USB. Here the USB will power the module, as well as send the signal for the shutdown of the computer. We will easily be able to detect whether the computer is on or off through the USB as well. The overall circuitry of the module will be different, as we will be using the USB for power, status detection, and status change. After some research, we found that the signal we have to send is slightly different between operating systems. We will aim for Windows systems, the signal to which is the USB Suspend signal. This is easily sent through the USB of the device. An idea of what this will look like with all of the components including the MSP:



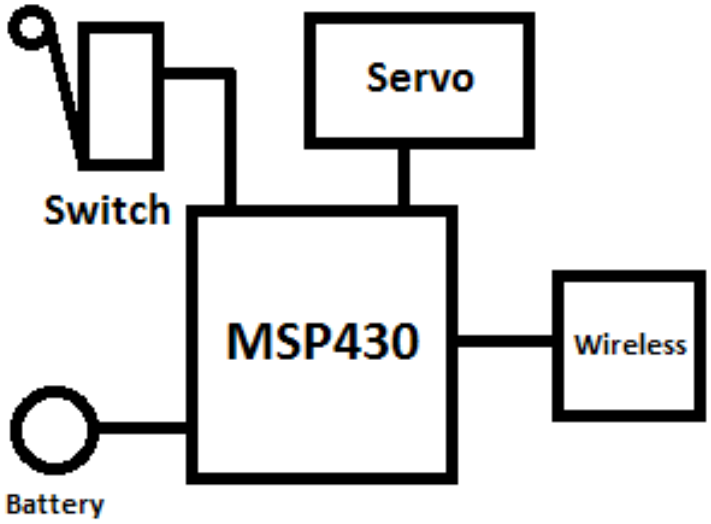
3.4.6 Lock Modules

The final module will be the module we use for locks. This will not be so much a modular adapter like the rest of the C2H modules will be, but rather an entire lock assembly replacement. We are not trying to reinvent the wheel here, so we will be using standard lock technology available to use now. In this module, we will have the wireless communication as an in/out, a status detection device, as an input, and a servo as an output to lock or unlock the door. This will be unlike previous modules, as it does not have an easily accessible source of power. We will be running this module on batteries. This is where the low power functionalities of the MSP430 come in handy. Since the MSP430 will spend most of its time in standby mode, the batteries will last for quite a while. We are planning on using two AAs to power this device. One of the unique attributes of this module will be the detection system. Since we cannot use the same method we are using for the light switches, which is a simple method of keeping track of the physical toggles of the light switch. In order to keep track of the status of the lock, we need some kind of sensor that can physically monitor the status of the lock. We narrowed our options down to two, either break beam eyes, or a micro switch.

Break beam eyes are sensors that send a signal while there is nothing obstructing the receiver or transmitter. When something obstructs the two, the signal is broken, and so is the circuit. One of the biggest problems with this solution, is the eyes must be aligned at all times. Slight misalignment issues can send off a false positive, thereby making the user think the lock is unlocked when it is really locked. The break beam eyes are also subject to a shorter lifespan than the micro switch option as it has current constantly running through it. The micro switches on the other hand, have a life span of 50,000 cycles on average. These lifespans are based on the mechanical tolerances of the spring inside the micro switch, as the internals are nothing more than a spring that completes a circuit, so the wear and tear is the same as any other wire. For the choice of micro switches, the only real difference is the weight in which the switch is tripped. The weight does not matter, as it will be situated behind the lock, so that when the door is unlocked and the bolt is retracted, the module will read the lock as "locked".

For the servo, we will need something small, so that it can fit inside of the lock panel, and inside the door itself. We will also need something with high toque. After doing some research on similar projects[8], we found a toque value of 51 oz-in to have have done the job. We also know that we are interested in running the Lock Module on a 9 volt battery, so any servo that can be powered in that range will do. I began searching at a website I have done business with in the past, and had an excellent experience with. The business is Futaba, and they make excellent miniature servos. After perusing their website, we found a servo with small dimensions, and who could run on 4.8 Volts which is well within our power supply range. Like previous modules, we will be using the CR2302 for RAM Retention. As mentioned previously, we are using a 9 Volt battery to power the Lock Module. We will talk more about battery life in section 5.7. The following is a picture of the general idea of all of the components connected together:

8 Texas Instrument's E2E SmartTouch. Printed with permission from Juan Garcia



3.4.7 Occupancy Sensor

One of the main goals for our project is to reduce the user's power bill by %20 a month. We plan to do this by setting up the ability to track when a room in the user's house is occupied or not. A state machine inside the board would keep track of whether or not someone was in the room, or the room was empty. Whenever a room is determined to vacant for a user-defined set of time, any or all of the lights, appliances or other power drains in the room can be turned off. We also want to be able to turn on any user-defined devices once someone enters the room as well. By doing this, we can greatly reduce the amount of electrical waste the user experiences. We will also want the user to be able to place these between any doorway of their choosing. This means that the devices would be self-contained, and not have to be directly connected to any other module. As you will see through our research, the power drain on some of these options is quite high. An option to overcome this is to wire the devices into in house wiring. There are of course a lot of sensors someone can use for this type of project, light, pressure, sound, heat, infrared and several others. We have listed in the following sections the options we have for this project.

3.4.8 Ultrasonic Sensor

Ultrasonic sensors use ultrasonic sound waves to detect whether there is an object in front of it or not. After the wave has been sound, the user would be able tell how far, if any, an object is away from the transmitter. The ultrasonic sensor in particular that we are examining is the MB1004 from MaxBotix. The MB1004 statistics are as follows:

MB1004	
Acquire Time	2.5 Second
Release Time	1.5 Second
Resolution	1 Inch
Current Requirement	2 mA
Voltage Rating	2.5-5.5V

We were able to narrow down the type of sensor we were hoping to use by first looking at the distance across which the sensor would work. Sense we are only doing this across doorways; we don't need a very large working distance. Fortunately, a lower working distance equates to a lower cost. Something else we are looking for is the acquire time. That is the amount of time it takes for the sensor to get an accurate reading on the object the user is trying to record. We can see from the table that the MB1004 has an acquire time of 2.5 seconds. Since this is a relatively long time, we would not be able to put this in a door frame as we had originally wanted. We could however, position this in the corner of a room, and take measurements periodically. In order to accurately determine if whether or not the room is occupied, there would have to be a period in which the device syncs itself. The user would have to position the sensor, leave the room, and tell the hub that the distance sensed is a "normal" value, and that any other value means that a person is entering the room. The obvious problem with this is that to accurately determine the moment someone walks into the room, we would have to constantly be pulsing the ultrasonic sensor. This leads to the obvious power consumption issues, as a steady pulse will drain the battery, and something that pulses intermittently means less accurate results. The table above shows that the MB1004 draws 2 mA every second that it is active. On a standard AA battery, which has a capacity of about 2000 mAh, a sensor would be able to run for about 1000 hours, or 41.6 days. This can be increased with additional batteries of course, although we would not want to add to many as this would make the device bulky and unseemly to the eye, and while it is not a main goal of ours, aesthetics play an important role in anything a user puts into his/her house.

3.4.9 Photoelectric Sensor

Our second option is for a photoelectric sensor. A photoelectric sensor uses a photocell to detect a beam transmitted by a laser or other optical light. The idea using this detection system would be to place the transmitter module on one side of the door and place the second module on the other side of the door way. As the beam breaks, the microchip would register, and we would know whether or not someone had passed the sensor. An issue that arises with just one laser, is we would be unable to tell whether or not someone is entering or leaving the room. What we have decided to do to remedy this, is setup a pair or photoelectric sensors. Depending on which sensor is tripped first, we can determine which direction the object is moving, either into or out of the room.

For the photoelectric sensor, we are looking at low power laser diodes to keep power draw low. After looking through various documents and websites for part listing, we have landed on the LC-LMD-650-01-03. The following is the data on this laser diode:

LC-LMD-650-01-03	
Power Output	3 mW
Voltage	3 VDC
Wavelength	650 nm

As you can see from the table, the output power of the laser is fairly low at 3mW. Since it is powered with 3VDC, this means that the laser module will draw about 1mA while it is active. Going back to our example of the AA battery with a capacity of 2000 mAh, this means the laser would be powered for about 83 days. This of course does not take into account the draw of the board, but it is relatively negligible compared to the laser pull. The problem with using a photoelectric sensor, is the easiest way to install this would be to use a wireless device, however for the scope of this project, we are not interested in creating an additional wireless receiver for this module. What we will be doing is running a wire around the door frame. With the way most house door frame are made, it would actually be very easy to pry the molding away from the door and run the wire, and it would look cosmetically pleasing. The wire run around the door frame would have

a photocell on the other end, which is what we would use to complete the circuit and gives us our reading. One main advantage we like with this option over the ultrasonic sensor is that the sensor is real time, the instant the laser is broken, we receive a signal, where as with the ultrasonic sensor, there is an acquire and release time to take into account. Another advantage is while the ultrasonic would detect anything in front of itself, the photoelectric sensor could be place at a height on the door so that only users can trip it, but anything lower than the beam, like pets, would not be able to trip the laser beam. Furthermore, as an option for power and wireless connection, we will easily be able to wire this to our lock devices we use on any doors that are in the area, as they are using AA power supplies themselves. This makes the overall installation and hub configuration of the system as a whole much easier. We feel like this is a good overall option for our project.

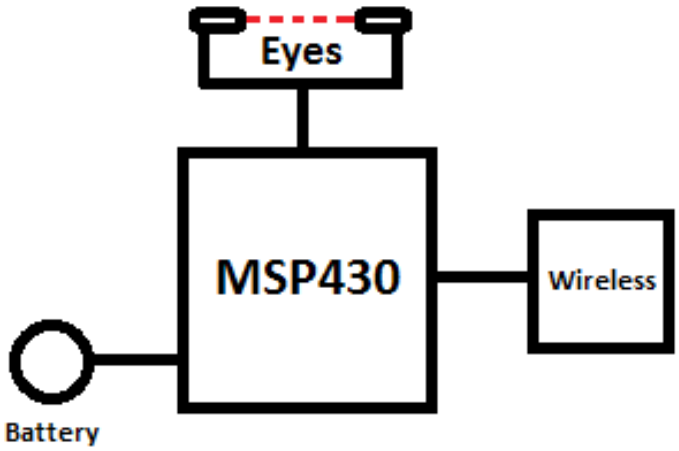
3.4.10 IR Motion Detector

Another option for the type of sensor we could use for our occupancy detection an IR Motion Detector. Essentially, a motion detector takes a “picture” of the undisturbed room, and sends a signal whenever the “picture” is disturbed. After some research, we found a motion detector that suits the scope of our project. The part number for this component is SEN-08630. This unit works between 5V and 12V, which is the approximate voltage that will be running through our boards at any given period. An issue we must revisit is the problem with realizing whether a person is entering or leaving a room, as we don’t want to turn the lights off when someone enters or vice versa. We can solve this problem the same we did with the photoelectric laser, by setting up two motion detectors. Motion detectors will pick up disturbance from wide view angle, in order to isolate the motion detectors so they are only picking up a signal at a time, we would situate one on each side of the door frame This of course will increase the power required to operate the sensor as we have two motion detectors running. Each motion detector runs at 1.6mA, so the two running together will drain 3.2mA while active. Deciphering the direction of the disturbance would be similar to what we do with the two lasers, depending on which motion detector senses the disturbance first; we can ascertain which direction the person is moving. We would simply use a finite state machine to keep track of how many people are entering and exiting the room, and changing the state of the lights in the room based on this number.

Another apparent issue with the motion detector method is that any object would send a disturbance to the motion detector and send a false positive. An example would be pets, fans, or even a sudden overcast could cause a disruption.

3.4.11 Sensor-Type Conclusion

With all of our options researched properly. We feel that the ultrasonic sensor is not viable; as it has the greatest chance for false positives with it's acquire and release time. This leaves us with the photoelectric and the motion detector. Based purely on accuracy, we feel like the photoelectric sensor is the best option, as we could place it high enough that pets would not trigger the sensor, and we obviously wouldn't have issues with in room disturbances like a ceiling or oscillating fan. Outside disturbance aside, the two motion detecors will use 3.6mA while active, while the photoelectric sensors will use 2mA while active. Since we are looking for the longest battery life possible, this solidifies our decision, and we will be working with the laser-based photoelectric sensor for our occupancy sensor. Since we have decided on the photoelectric sensor, we offer an idea of the components involved here:



3.6 Circuitry Housing

As far as what we will do with the circuitry once we have completed it, we will need someplace to put the board to keep it from shorting, or from outside disturbances. Unfortunately, we do not have access to any high-end injection molding service, but we do have a few options we can do with the resources available to us.

3.6.1 Alumilite

The best option for something like this would obviously be injection molding plastic to fit our boards exactly. Unfortunately, this is not something you can do without a lathe, which we do not have access to. It also requires quite an initial investment for the first plastic piece you make. What we can do is try the next best thing, and that is resin. Setting resin is actually very simple, although a bit time consuming. The process involves creating a mold, and filling the mold with resin, which depending on the type of resin can set as hard as plastic. One particular brand of resin is Alumilite. It is a two-part resin, that is, it begins as two chemicals that react to one another once they are mixed. To create the mold, you begin by creating two halves, which hold the shape of what you're trying to make. Generally, resin is used to make copies of objects. The upside to this, is that you are able to use the original to make an imprint in the mold, making it easy to replicate very complicated pieces. Unfortunately, we do not have an original piece, as we are creating everything from scratch. Fortunately, the shapes we are looking to make are quite simple. Making the initial molds can be done with clay, and a steady hand. After the two halves of the mold are made, you mix the two chemicals, and poor the resin into your mold. After setting for a few ours, the plastic piece is then ready to be used. It is even possible to color the resin, so that it is possible to make our modules pleasing to the eye.

3.6.2 Salvaging Similar Casing

Another option for finding plastic housing is salvaging similar casings from other products, and milling them to fit our needs. As you will read in section 5.10, there are essentially three unique housing we need to make for our modules. For the most part, since we are trying to keep

our designs simple, the casings are just rectangular boxes. It is actually very possible to find similar products which use these simple shapes as their casing. With the help of a dremel, we would be able to modify these existing casings to fit our need. The down side to this is obviously the aesthetic value, as these would not be very pleasing to the eye. The other downside is losing the uniqueness of our product to something that has already been created. We also would open ourselves to potential legal action depending on the types of patent the manufacturer has, as sometimes the case itself is patented. We also run the risk of not finding anything at all, due to the placing of screw hole possibly drilling through our board. With all of this in mind, we believe we will forgo this option to find and option with much more customization.

3.6.2 3D Printing

When it comes to methods of creating plastic shapes customized to a specific purpose, a lot of people are now thinking 3D Printing. 3D Printing is an option that has just recently become available to the general public. Fortunately for our group we have access to a 3D Printer via the professional engineering fraternity Theta Tau. We would be using a RoBo 3D printer, which uses PLA plastic. One limitation of 3D printing, is that we are limited to the space on which the 3D Printer prints, and the RoBo 3D is actually relatively small compared to other 3D printers. However, our plastic casings are generally small (the largest being 55mm x 70mm x 20mm), so we will have no problem here. We have also mentioned several times now that we are looking for a somewhat aesthetically pleasing look to each of our modules. This is made possible by creating cases that are made specifically for our hardware, as there will be no extra holes or pieces hanging from our prototype as there would be if we salvaged parts from similar shaped plastic. One downside to using a 3D Printer is the amount of time it would take to print out the casings we need. While some of our pieces would be rather small, it would still take several hours for each piece to be printed, and we plan on printing several pieces. This can add up to days of printing time, which is much longer than either of the previous options for the housing. Ultimately, we believe that the customization and the thrill of working with a new technology such as 3D Printing, makes this our choice for our project.

3.8 Wireless Communication Dongle

We chose a wireless transmission method over a wired setup for various reasons. For one, if Close to Home were to require a lengthy procedure of installing wires within the house's already existing construction, it would deter consumers from wanting to use our product due to the large initial setup fees likely to result from such labor. As well, since the topology of the network mostly involved several smaller units performing burst communication in a scheduled manner, having a constantly-connected network of wires would be superfluous to the design. It's not necessary to always have a connection to the other units—only when the hub needs to override a setting on the device or hear a report from it on its environment, thus a wireless communication system is ideal.

The choice for what form of wireless communication that was appropriate for the Close to Home system was something that took up a large aspect of the research portion of the project. We wanted a communication medium that was not a large power sink, so that we could keep the overall power impact of the central hub down as far as possible. As well, we would need a communication method that would be easily encrypted—such that not just anyone could join your Wi-Fi network and spoof packets to the device in order to open and close your doors at their will. Lastly, the wireless system must be powerful and reliable enough to send packets of information with over a 90% success rate, and perform this transmission in a timely manner enough to keep up with the scheduling properties of the Close to Home hub unit.

3.8.1 Bluetooth (IEEE 802.15.1)

One of the initial choices for a wireless communication mode was Bluetooth, as it is a fairly easy to set up architecture that can yield great communication over short distances. The maximum reliable transmission distance of a pair of Bluetooth devices is about 10 meters. IEEE 802.15.1 was created to replace wired USB (Universal Serial Bus) connections for devices within very short range of their destination, such as a keyboard or mouse next to a computer. Bluetooth's expertise shines in its usage of a collision algorithm known as FHSS (Frequency-Hopping Spread Spectrum), which avoids collisions in data by blasting transmissions across multiple frequency channels at a very rapid rate.

By synchronizing the transmissions across certain frequencies and not across others, collisions are avoided or at the very least greatly reduced.

This makes it perfect for a Close to Home application in a smaller apartment or a room-by-room configuration, but the 10 meter range quickly becomes an issue when dealing with any normal-sized house or environment. While its true that Bluetooth could be used in a sort of ad-hoc configuration, allowing transmissions to hop between nodes, any two nodes could be no farther apart than 10 meters, adding on an annoying limitation that would be a hassle for the group and consequently any potential consumers, to deal with.

Bluetooth	
Standard	IEEE802.15.1
Data Rate	1 Mbps
Communication Distance	10m, 100m
Power Consumption	120mW
Lifetime	Several Weeks
Number of Nodes	7
Security	64 bit, 128 bit

3.8.2 Wi-Fi (IEEE 802.11)

The 802.11 architecture of wireless transmission is one that has a plethora of support. Not only do all home internet routers utilize the 802.11 form of transmission (or its evolution of 802.11b and 802.11g), it also has a great built-in security support in its packets in the form of WEP or WPA-PSK. However, since the Wi-Fi architecture is in use throughout nearly every home, there can be interference issues if we were transmitting data that was necessary for Close to Home on the same frequency as the one used for the home wireless network. 802.11 has relatively powerful transmission power, allowing it to travel up to 110 feet without the investment of any kind of high-powered antennas. On top of that, transmission through walls can be done with relative ease, and with minimal loss in signal strength and degradation.

A very small USB device could be plugged into a device such as the Raspberry Pi to provide it with the ability to transmit over this protocol,

and it would be rather affordable, at no more than $10 for the unit. Though this is true for the central unit, it would be necessary to outfit each monitoring unit with the ability to transmit over the 802.11 interface—something that would prove to be a costly endeavor, considering the price of the hardware. As well, the Wi-Fi architecture is nothing, if not power hungry. Utilizing the 802.11 transmission protocol within each monitoring unit within the home and inside of the centralized hub would raise the power taxation of the Close to Home system to a point that could very quickly outweigh the money saved by installing the system in the first place. Since this is obviously against our initial design for the product, we needed to opt for a system that was lower in power consumption and had a lower price point.

Wi-Fi	
Standard	IEEE802.11a, b, g
Data Rate	11 to 105 Mbps
Communication Distance	10 to 100m
Power Consumption	>256mW
Lifetime	Several months to several years
Number of Nodes	65535
Security	Various

3.8.3 ZigBee (IEEE 802.15.4)

The final transmission protocol that was considered for Close to Home was the IEEE 802.15.4, also known as ZigBee. This protocol is a low cost, low power, and low transfer rate standard for wireless communication. ZigBee was initially developed for a purpose much like our own—the communication of multiple small, power-conservative units that are in place for a specific monitoring or maintenance purpose. The protocol is very reliable, utilizing a handshake method for communication, and sacrificing some of its transmission speed for increased packet consistency and length of travel. The typical range of a ZigBee communication is about 50 meters, which should be substantial for a medium-sized home, and will transfer without line of sight without issue. The data transfer can also occur on a varied band of frequencies, from 2.4-2.4835 GHz, 868-870 MHz and 902-928 MHz. These three bands were sufficient enough for the Close to Home system to run into very little interference from other household devices.

Keeping our data secure as it moves across the house is also a primary concern, considering the large amount of security features that Close to Home will utilize. Luckily, the ZigBee protocol also has the ability to be easily encoded with our own custom form of cryptography. There exist forums and guides online that explain the entire OSI stack of the ZigBee architecture, and finding what part we can safely encrypt and decrypt was a simple matter, allowing us to have full control over the amount of peering eyes we have into our work, and whether or not a rogue user can listen in on our transmissions.

Another of the great aspects of ZigBee is the ability for the devices connected to a ZigBee network to only have intermittent communication with the hub unit. This means a very long battery life for the monitoring units if they are intended to be run upon battery, or extremely reduced power consumption if the monitoring units are connected directly to the house's power grid. This comes at a cost, however, due to the fact that ZigBee transmissions involve an entire OSI stack, resulting in packets with significant overhead and transmitting much slower than the other protocols, such as Bluetooth and Wi-Fi. However, since our smart-home application requires very little speed in transfer, and only intermittently so, ZigBee seems like the ideal fit. We will explain the hardware capabilities and the topology we chose in the further sections on the Wireless System.

ZigBee	
Standard	IEEE802.15.4
Data Rate	20-250kbps
Communication Distance	75m
Power Consumption	60mW
Lifetime	Several months to several years
Number of Nodes	65535
Security	128 bit AES

4.1 Android Application Design Details

The Android operation system is a versatile and lightweight system that we will use to achieve all our intended designs for the overall system. It provides us with enough tools to complete our proposed operations, system requirements, design, and testing for completeness and development.

4.1.1 Concept of Operations

In this section we discuss our concept of operations on a high level. We had to look at why our application would be useful in the system and for whom it would be useful for. We address as many operational scenarios and features we want our system to accomplish and how to implement them.

4.1.1.1 Proposed System Needs

The Android application for our Close to Home (C2H) system is meant to make the
user interaction with the entire C2H system quick, easy, and natural. The
application provides streamlined access to all components of the system the user needs to interact with such as light status, door lock status, detecting and setting temperature, and monitoring power consumption of appliances. The main goal for this application is to make performing these task easier and more convenient for the user than if they were to do it manually. It should also provide the user with a sense of mind that they are in control of their house wherever they are at and are able to save energy and money by having such control. The application itself is the main graphical user interface for the system.

4.1.1.2 Users and Modes of Operation

The application has two classes of users:

Head of Household: This user will have full functionality of all of the systems functions. They will have the ability to set all device statuses. In addition they will have the ability to set the permissions of other users in the household. These other users will be called “Tenants”

Tenants: These users are the subclass to the parent class Head of Household. They are able to control whatever device statuses the Head of Household allows them to have access to.

There are two modes of operation that the application can be in. The modes are based of of the current data connection. If the Android-powered mobile device is on the same wi-fi as the hub and devices it will not require a secure log in as it will be able to communicate with the hub directly through the wi-fi network. If the device is on mobile network connectivity it will require a secure login and need to communicate with the database so that the hub can periodically check the database for status changes made by the application.

4.1.1.3 Operational Scenarios

Our system is meant to provide the user with an easy and natural way to operate and monitor devices and appliances in the house both in the home and away from the home. One of the most beneficial scenarios of our system comes from the energy saving viewpoint. Say the user leaves the house and does not remember if they turned off their lights in the living room, or an even more hazardous condition, they left their oven on. With a simple log in to our application they are able to shut off the lights and save money. While we don’t intend on making a power shut off option in this system model, monitoring the appliances like the oven can at least give the user a sense of mind that their home is safe or if it was left on, give them time to take appropriate safety actions. The application will always be in hand wherever they have their phone. With our application out of home they have the most power saving abilities.

Our system is limited in the situation of needing power to run the system as well as internet connection. This can lead to some security issues since our system will also control door locks in the house. The best situation is to make sure there is always a backup battery in the hub so that it can still control the devices in case of a power outage. The hub will also be in charge of putting all of the status monitoring

devices into a manual mode in case of a power outage or loss of internet connection. The manual mode can be defined in the app in the Head of Household user profile. There they will define which devices are turned off or locked/unlocked in case of power outage or internet loss.

4.1.1.4 Operational Features

Features:

- Light Status (on/off)
- Door Status (locked/unlocked)
- User profile settings
  - Head of Household
    - Ability to set Tenant user permissions.
    - Ability to set manual mode defaults in case of power or internet loss.
    - Full control over all other features.
  - Tenant
    - Full control over whatever features are not restricted by the Head of Household.
- Device battery status so we know when it is time to change the devices batteries.
- User login for when user is not connected to the wi-fi the system is implemented on allowing for mobile applications.

Possible Features:

- Voice commands
- GPS location

4.1.1.5 Implementation

The application will be developed on the mobile Android platform which uses Java as its development language. It will be developed in the Eclipse workbench using various Java libraries for the java sdk. The application will require wi-fi and mobile network connection in order for it to work. It will need to communicate with the SQL database as well as the hub. From there it can send the data from the user input to the application. Then finally send the status data to the device. It will also be able to read status data from the database and the hub in order to view device battery levels and status on the users device running the application.

4.1.2 System Requirements

In order for our application to function with our objectives and goals for the entire Close to Home (C2H) system it must adhere to the following requirements.

4.1.2.1 Software to be Produced:

The C2H software is a communication gateway between the user and devices in the users house. It is meant to make interacting with the house quick, natural, and convenient. The GUI is streamlined and easy to access and understand. The basis is that the application allows users to control and monitor devices in their home. Underneath the UI, the application communicates with the main hub module of the C2H system as well as the database.

Use Case Diagram :

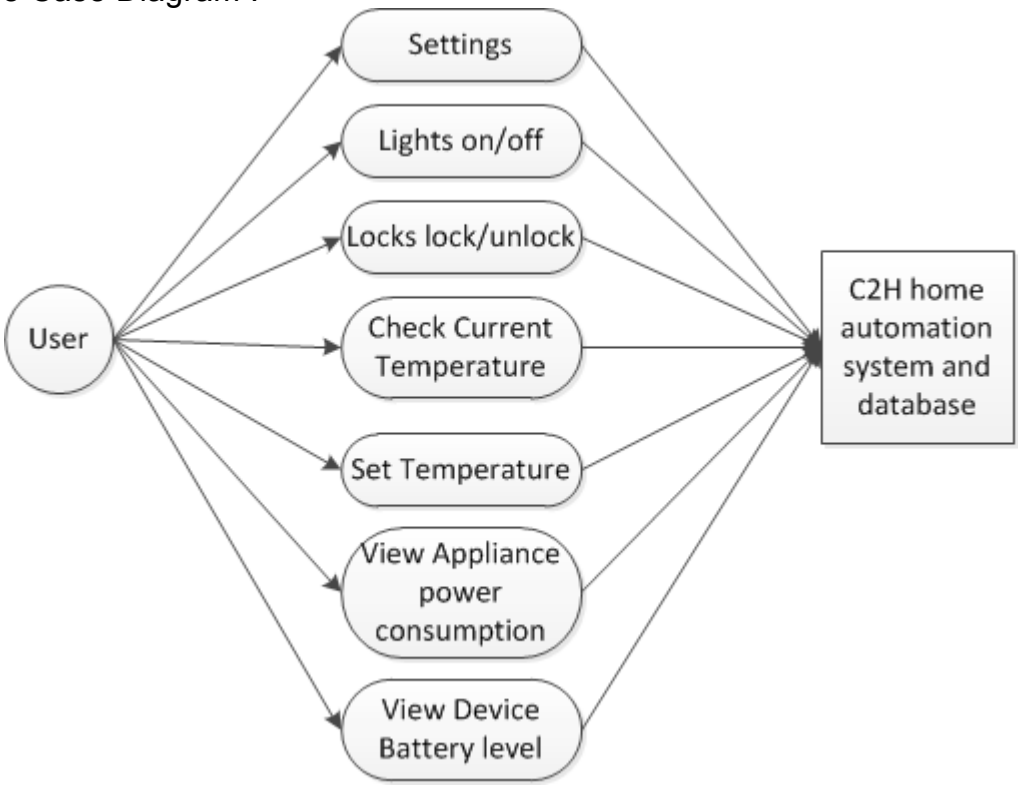


Fig 4.1.2 Use case diagram showing users input and uses for the application. Once the user inputs the data it is sent to the rest of the C2H system.

4.1.2.2 Functional Requirements:

- User verification
  - Login required if on mobile network
- Home Device Functions
  - Set various lights with unique names to either the on or off status and display status to the user.
  - Lock and unlock various doors with unique names and display current status of the lock
  - View appliances with unique names current power consumption to help the user know if the appliance is off or on.
  - View each devices battery status so the user knows when it is time to change the devices batteries.

4.1.2.3 Interface requirements:

- Inputs:
  - Login, Settings (permissions and manual mode defaults), device status, Air conditioning temperature.
- Outputs:
  - Users selected device status/temperature, login data
- Datatypes:
  - Device status, Temperature, Power consumption, battery levels
- Accuracy of Data:
  - Lock and Light status boolean on or off
  - Temperature to the nearest degree
  - Power consumption to the nearest tenth of a watt
  - Battery levels to the nearest ones place in percentage
- Frequency of Events:
  - Device status,temperatures, power consumption, and battery levels will update and check for changes every time the device is selected by the user.

4.1.2.4 Physical Environment Requirements:

The software will run in an environment wherever the Java powered mobile device is limited to running. The Close to Home system will need to be fully installed to the desired home and network in order for the application to be effective.

4.1.2.5 User and Human Factors Requirements:

The users will be any person looking to have their homes automated by the C2H system. The user should have basic knowledge on how to use their smartphone with the mobile application installed. (if time allows, a user manual with tutorial will be created)

4.1.2.6 Data requirements

Lock and light status will be sent as boolean. Temperature and battery levels will be represented by integers. Power consumption will be represented by floating point integers.

4.1.2.7 Resource Requirements

Team of four programmers able to create, troubleshoot, and maintain the application in Java is required. Time spent developing this application is split between the four over a one semester time period (approximately 3-4 months) and must be factored into the the other components of the Close to Home system time constraints. As for funding, the funding is limited to the sponsorship funding given by Duke Energy of the amount of $1,800 U.S. plus any personal funding from the team members. The funding is also split up into various components of the system.

Software required to develop this application include a computer with an operating system running the Eclipse IDE as well has have the Android sdk installed for mobile application development. A SQLite compatible workbench is also required in order to develop and manage the queries made by the application.

Hardware required is limited to android devices running 2.x to 4.x version of Android available for testing and development.

4.1.2.8 Security Requirements

The application must be secured by two ways. One over the Wi-Fi network and the other over the mobile network. A user should not be able to access the inner functions of the application if the user is not connected to the same Wi-Fi the Close to Home system is connected to. This leave the security to the Wi-Fi's level of security. Suggested use of a secure WPA wireless connection. When connected via mobile web the application should not let the user access the inner functions of the application until the correct username and password is input into the login screen. This data is encrypted on the web server and will only grant permission to communicate to the home system that belongs to that particular login.

4.1.2.9 Quality Assurance Requirements

Reliability:
The application will be designed to be lightweight so that many different devices with different operating power can run the application at anytime so long as the device itself is reliable.
Availability:
The functions of the software are only effective with an internet connection via Wi-Fi or mobile network. The rest of the Close to Home system must be running and powered as well.
Maintainability:
The application software can be maintained via updates to the software. For our purposes, the installation and updates can be done locally and on a case by case basis. If there were to be a mass production, we would have to develop a support team and push updates to Googles application marketplace that manages update releases.

4.1.3 High Level Design

The way the user accesses the inner functions of the application depends on the connection the users device is using whether it be Wi-Fi or mobile network. If the user is on the same Wi-Fi connection as the rest of the system, then the user has access to the home device

functions and can make various operations such as turning on or off the lights. If the user is on a mobile network connection and they open up the application, they will need to enter in their login information and the application will check with the secure database if the login is authentic. After authentication, the user is then able to access the home device functions. If the user is the Head of Household user (see 4.1.1.2 Users and Modes of Operation), then the they will have access to a settings menu where they can set up the defaults for manual mode (in case of network or interuptions) and Tenant user permissions. All input data into the home device functions module is then sent to the main C2H hub where the hub performs the desired operations to the external devices. The device status is also periodically updated in the database from the hub.

Formal Diagram of High Level Design Hub:

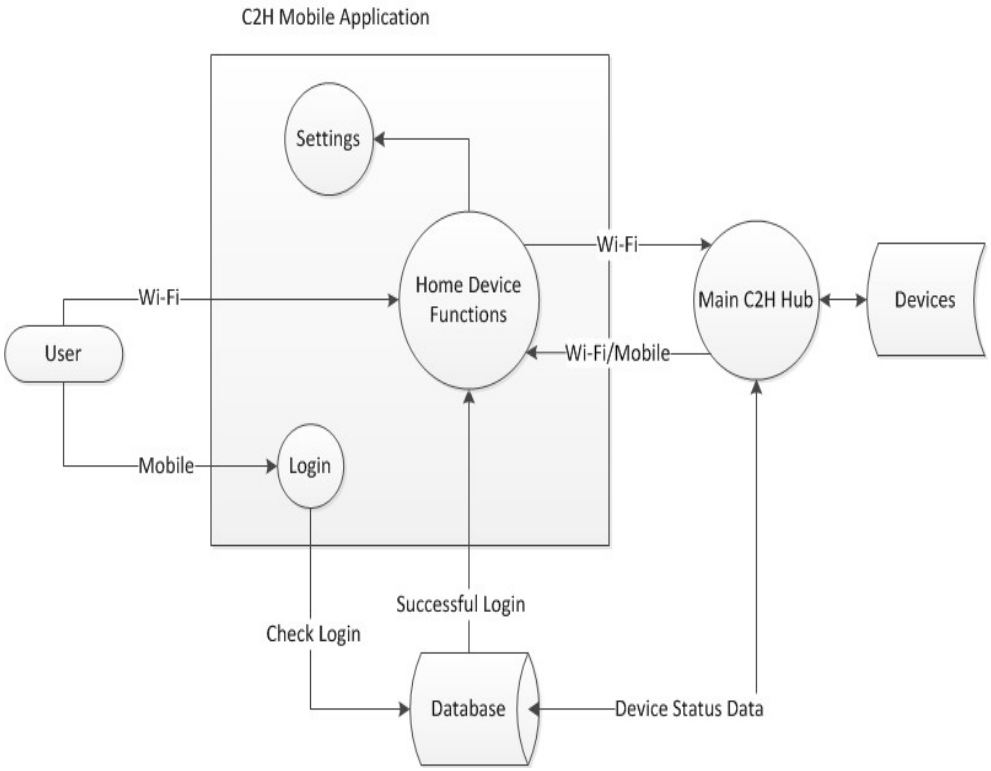


Figure 4.1.3 shows the high-level architecture of the applications system and its interface. Arrows represent data flow between components of the system.

Our design is meant to be highly reusable as it is considered to be a real time application that can be adapted to any home with wireless internet connection and access to an android device. There are drawbacks in that design such that a house with no android devices and/or wireless internet connection cannot use our system.

As for maintainability, on a high level the software has few components. This makes updating aspects of the design relatively easy for the developers or any further support with Android application design experience. Also, with the right amount of resources, our system can be easily updated to accommodate any new home configuration. This simplicity is part of the goal of creating an easy to use interface for the user so that there is some benefit to using our application.

The design will be tested to ensure that each of the respective functions on a high level complete their intended purpose and do not conflict with any other parts of the system.
At a high level, the software is easy to test as the various functions do not have too much dependencies with other functions or other parts of the systems. That parts that do have dependencies can be tested easily using dummy input and output data such as the data coming back into the application from the hub or authentication coming from the server.

Performance is a main goal in order to obtain a beneficial outcome for using the application. If the performance is low, the application becomes less valuable to the user. The application is designed to be lightweight and simple therefore it will run with a stable performance on most devices. Using the appropriate Android sdk libraries also ensures we are not overstepping the boundaries of performance for a given OS version.

As far as portability, the system will be available on any device running a supported version of the Android OS. Another one of the main goals for this application is that portability is key. This is why we implement the database component so that the user can communicate with the system wherever the user has web connection.

There are a few safety concerns. We don't want outsiders to have control of the locks or other components in your home. In order to make the application secure, it is kept so in one of two ways. The first is already implemented by the users wireless home network. For our purposes we will suggest and test the system on a secured wireless network. For the second attempt at keeping the application private and secure we implement a login that will identify a user's network and keep others out of that network with an encrypted authentication to the server. There is also the implementation of different classes of users. One is the Head of Household who has full control of the applications functions, and the other is a Tenant who only has control over functions specified by the Head of Household. This is sort of a 'parental control' so that users, such as kids, cannot control devices that the head feels are not appropriate such as temperature or specific locks.

We will have an evolutionary prototype that will perform system requirements to ensure that the system is fully functional. Since we will be using an evolutionary prototype, our designs will change as we find additional or excessive/non-working functionality. We decided to use an evolutionary prototype rather than a disposable prototype because it makes more sense to build off of what is already functional.

4.1.4 Low Level Design

In this section we discuss the application in more detail as well as address any design issues we encountered in the high level design 4.1.3. Further we Trace through the design requirements we set for the project and how they are implemented in the detailed design of the application.

4.1.4.1 Design Issues

We designed the Close to Home application to be highly reusable as it is considered to be a real time application that the user can access anytime anywhere. Unfortunately there are limitations that can cause our application to not be as reusable as we would like it to be. Two main limitations include the operating system we choose and the wireless internet connection requirements needed for our system to successfully run. Our application will only run on Android devices running operating systems 2.x to 4.x versions. The application will only run with internet connection as well. We decided to forego the option to bypass the database login when connected to the home wifi network due to complications in consistency and redundancy. It would take more time and power to update a local database as well as a hosted database therefore all data for the house will be hosted and accessed in the same place. This makes it so that multiple users will have the same data and there are no conflicts while using our application.

In order to maintain our applications system we would have liked to have a simple to install system where once the devices are set up in the home, the application will identify the type and number of devices connected to the system but for early prototypes this will not be possible. As a design tradeoff, we want to ensure the system works for a predetermined size home with a set number of devices. these devices will be identified and hardcoded into the application itself until we have time to make the application more dynamic and able to update itself with more or fewer devices.

As far as portability, the system will be available on any device running a supported version of the Android OS. Another one of the main goals for this application is that portability is key. This is why we implement

the database component so that the user can communicate with the system wherever the user has web connection.

There are a few safety concerns. We don't want outsiders to have control of the locks or other components in your home. In order to make the application secure, it is kept so in one of two ways. The first is already implemented by the users wireless home network. For our purposes we will suggest and test the system on a secured wireless network. For the second attempt at keeping the application private and secure we implement a login that will identify a user's network and keep others out of that network with an encrypted authentication to the server. There is also the implementation of different classes of users. One is the Head of Household who has full control of the applications functions, and the other is a Tenant who only has control over functions specified by the Head of Household. This is sort of a 'parental control' so that users, such as kids, cannot control devices that the head feels are not appropriate such as temperature or specific locks.

4.1.4.2 Detailed Design Information

This application is designed to interface with the database which will act as a relay for data the user inputs into the application to reach the main hub. From there the hub will control all devices wirelessly.We plan to incorporate the entirety of our home automation system into one android application. Below is the class diagram for the application design.

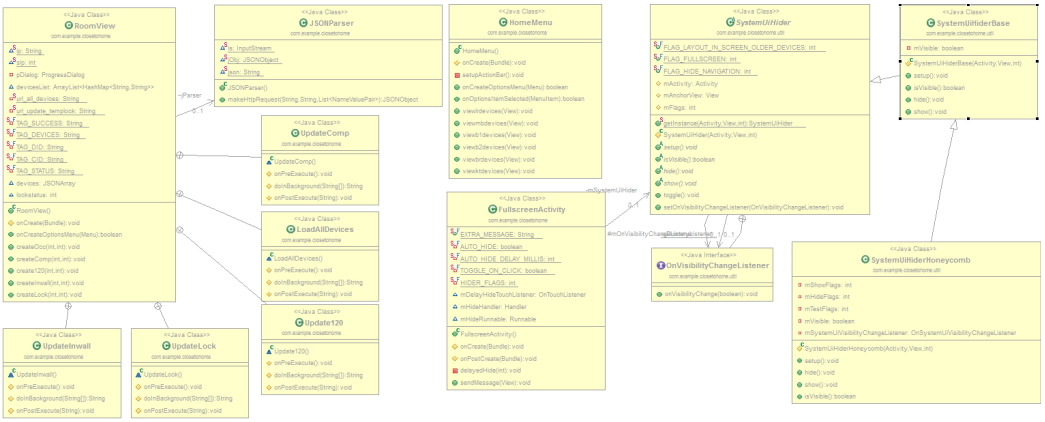


Figure 4.1.4.2 Close to Home application class diagram. Shows all of the classes with operations.

4.1.4.3 Trace of Requirements to Design

The C2H software is a communication gateway between the user and devices in the users house. It is meant to make interacting with the house quick, natural, and convenient. The GUI is streamlined and easy to access and understand. The basis is that the application allows users to control and monitor devices in their home. Underneath the UI, the application communicates with the database of the Close to Home system. That data is then relayed to the main hub where the hardware receives the data and the operations on the devices are carried out. Below is a list of requirements set for the application and how they are carried out in the Android application.

- User Verification
  - User must sign in with an authentic username and password in order to access the rest of the applications functionality. Once the application starts up it will lock the user in the login screen until the users credentials are verified on the database. Once the user is verified, the main menu will load up with that users permissions for that specific home. There will be only one Home per user. If the user wants to use the system for multiple homes, multiple accounts will need to be made. The MainFrame class will handle queries to the database to verify the users login.
  - The User class itself contains three attributes that can be used to verify and login the user.
    - *permission*: A boolean datatype that represents one of two classes of users. 0 will represent Tenant and 1 will represent Head of Household
    - *name*: This is a basic string that the user will type in at login. The string will then be checked in the database for existing users with the same name. Once an instance of the application is logged in, the name will remain the same for the entire session until the user logs out of the application.
    - *password:* This is a basic string that will be checked along side the users name to verify and authenticate login. Once an instance of the application is logged in, the password will remain the same for the entire session until the user logs out of the application.

- Home Device Functions
  - Once the user is logged in, the main screen is brought up. Each login is associated with one Home class. Each home can contain any number of instances of the class Room, which subsequently holds any number of instances of the class Device. Home Device Functions are unique to each Home class. The Home is identical for all users of the home with some permissions revoked for Tenants if a Head of Household makes those permission changes. The homes devices are all unique to that home. Included in the home class are four attributes.
    - *rooms:* This is an array of type room, that divides up the devices further into corresponding rooms in the house. Here is also where rooms can be added or removed from the array.
    - *homeName:* Is a simple unique string identifying the home system and all its entailing devices. Each user has a home name associated with it in the database. This is how the GUI will decide which data to access and which rooms and devices to load on login.
    - *connected:* This a boolean that regularly checks if the hub is connected to the internet. If not the boolean switches to let the user know there is no connection to the internet for the hub. The hub will store default settings set by the user that the devices will switch to in case of internet loss or power loss.
    - *users:* This is an array of type users that hold all users that have access to the House devices. This is how the database relays its information on the users and their permissions. Users are added and removed to and from this array.
- Room is a subclass of Home that helps divide the the devices for easier manageability as well as fast and fluid user control.The class Room has three attributes managed by the application and database.
  - *occupied:* This is a boolean that is set as 1 if occupied or 0 if vacant. This data is sent from the database since the input comes from the break beams and not user input to the application. For each room, default settings for the

devices can be set for when a room is occupied or unoccupied.

- *devices:* This is the array of type device that the application uses to display all the devices for the specified room. The data is pulled from the database to indicate which devices and what type are in each room.

- In order for the user to control specific devices, the devices are organized into each room. From the home screen the user can select a room and then view an easy to navigate and read list of devices in that room. The devices are all identified by two attributes that are used by the database to store and make status changes to the device.
  - *type:* The type int will tell the database and the GUI what type of device it is as well as what parameters to display for the device.
    - 0: Light
      - A basic on/off button labeled with the device name is available to the user if they are permitted as either a head of household or the permission is not revoked by a head of household.
      - Battery level status is also displayed next to the device name.
    - 1: Lock
      - A basic lock/unlock button labeled with the device name is available to the user if they are permitted as either a head of household or the permission is not revoked by a head of household.
      - Battery level status is also displayed next to the device name.
    - 2: Temperature
      - An integer will be displayed showing the current temperature recorded by the device. A combobox filled with a range of temperatures from 50 to 100 degrees fahrenheit in increments of 1 degree is available to the user if they are permitted as either a head of household or the permission is not revoked by a head of household.
      - Battery level status is also displayed next to the device name.
    - 3: Appliance

    - A floating point integer is displayed showing the current power consumption in Watts to the nearest tenth of a Watt. Due to the vast array of possible technologies and complexity in creating custom controls for various appliances, only one option will be available for appliances which is the Kill option
    - *Kill* will allow the user to cut the power to the appliance with the click of a button. For accidental shut off prevention a confirmation screen will pop up in order to confirm that the user wanted to shut off power to the appliance. This is useful for emergency situations such as forgetting to turn off the oven or energy wasting appliances like the television.
- The outputs are the user selected device statuses. They are all sent to the database via queries that update the data stored for each device.
- As for frequency of events, we want a fast responsive application that feels natural to the user and makes our application useful. If it takes longer for the user to perform the device task with our application than it does manually, then our application isn't effective. Therefore, we would like our device to update the database with current data as often as possible but we also must take into account data usage, computing power, and energy usage. For our purposes, updating the applications data as well as the databases data once every 40 seconds or every time a change is made to a device status should fit our goals for optimal performance.
- Security is important when controlling something as personal as a home. For these reasons our database will be secure and encrypted. As far as the application is concerned, the security is user based. This means that their home can be as secure or insecure as they want it. A Head of Household has the ability to create other heads of households or as many Tenants they want to. and tenants will have whatever access to the devices the head wants to allow. If the User is logged in as a tenant, options will only be available if the have the permissions. There will be a requirement of a password of at least 6 characters in length and at least one uppercase letter and at least one special character. This will help ensure the users account is well protected from password guesses or algorithms.

4.1.5 Test Plan

Testing this application is vital as it is a main component in the Close to Home system. The application is the users connection to the home system as well as a security wall between outsiders and the desired user base. Luckily Android makes testing easy as we are developing the application. Testing will take an evolutionary approach as development takes place as well as making sure individual test cases produce our desired results. Eventually the application will take a full prototype test running with the complete system on an android device.

4.1.5.1 Test Environment

We will develop the application in Eclipse with the Android Developer Tools (ADT) that provides an integrated development environment where we can create, build, and run Android application test cases from the GUI. From here we are able to run a test case for each activity and UI component individually. Further there is an Android Virtual Device (AVD) where we are able to run our application in a virtual device on different platforms so we can see how to best layout and optimise our application. As for testing output, we will need to have a local server with a database workbench in order for us to verify the output data is working as intended.

4.1.5.2 Test Cases

Below we list the test cases for the functionality of each class as well as the UI in general.

Class	Test Description and Methods
User	• Test to make sure a single instance of User is created when application launches • Make sure permission boolean matches up with correct permission status • Test that username and password matchup with that of the login user and database data • Test that each user is unique and only one user in the Home can have the same username • Method getPermission() returns the accurate permission boolean • Method setPermission() returns the accurate permission boolean and getPermission() returns the updated boolean that matches that in the database • Method getName() returns correct username string and matches that in the database • Method setName() returns correct string and updates the users username which can be tested using getName(). • Method getPassword() returns correct password string and matches that in the database • Method setPassword() returns correct string and updates the users username which can be tested using getPassword().
Home	• Test to make sure a single instance of Home is created when application launches.

	• Make sure the homeName string matches that connected to the user currently logged into the application in the database. • Method addRoom() adds another room to the rooms[] • Method removeRoom() removes a room from the rooms[] array • Method getRooms() should return an array rooms[] with the correct rooms for that specific house and no extra rooms that have been removed. • Method getHomeName() should return a string that matches that of the home name in the current session associated with the user and database data. • Method setHomeName() should change the home name associated with the current user as well as all other users associated with that home in users[]. The data should match in the database. • Methods getConnected() and setConnected() test that the connected boolean is 1 when connected and 0 when unconnected. (requires dummy data in the database until connected to the hub) • Method addUser() should add a user to the users[] and take in a permission boolean, name string, and password string. • Method removeUser() should remove the specified user from the users[] associated with that home. • Method getUsers() returns the current array of users[] that should match up with the desired users in the database.

Room	• Test to make sure the number of rooms created on startup of the app matches that of the rooms[] array as well as that the roomName strings a devices[] all match up. • Methods getOccupied() test that the method returns the correct boolean that was set by setOccupied(). • Methods addDevice() and removeDevice() make sure the correct devices are added to devices[] and removed. Database should match. • Method setRoomName() setst the roomName string and getRoomName() should return the same string as well as update the database. • Method getDevices() should return an array of all the devices associated with that room in the houme associated with that user.
Device	• Make sure the type field sets all other parameters to null except for those associated with that type int. • Make sure all the get methods return the correct status changes and all the set methods change the correct statuses and relay those changes to the database.
Main Frame	• Make sure all windows and frames render correctly. • Test that the correct device controls are locked for Tenants and Head of Households are able to manipulate the correct parameters and change settings. • Make sure this class runs the main and that it only generates one instance of home later to be populated after login. • Make sure the login screen is the only screen visible until the user is verified.

4.2 Possible Webserver Architecture

Based on the requirements for our project, WebIOPI will be too simple of a web architecture to utilize. Instead we will opt to create custom PHP scripts that will access our populated MySQL database via an Apache Webserver that we will establish.

In order to utilize PHP and Mysql on an Android application, we will need to start by renting or establishing an Apache Webserver.

4.2.1 Apache Webserver

This is a public domain open source web server. This means that that it is available for anyone to use, modify, and re-publish under the same circumstances. This has a few implications for our project: there will be lots of already done applications of Apache webservers and there will also be tons of resources to reference when crafting our Apache webserver.

Server space can either be rented from a private corporation or it can be established by turning your computer into a server with the appropriate software. Servers allow any of our data to be accessed over the internet which is the only way to relay information to our Android application when it is not at home. We could essentially program the entire application to work locally without a webserver, but this would eliminate half of the security and automation features that we want.

Once you have a web server established its time to setup your Apache webserver. Apache webserver is simply software that you use to gain certain functionality from your webserver like security and integration with other software like MySQL. Asimple guide to setting up the Apache Webserver can be found here: [9]

The Apache webserver can be used to authenticate our users, provide security encryptions to our transmissions, allow our Android application

9 https://www.dropbox.com/s/zxtvgi2ifg88h2d/www.dedoimedo.com-apache-web-server-lm.pdf

to reference the status of our system, allow our Raspberry Pi to update the status of our system over the internet, and to send commands from the Android application the the Raspberry Pi.

4.2.2 MySQL Database

MySQL is software for managing and utilizing a database system. This is necessary in order to provide access to our local data over the internet to our Android application. Our smart hub will operate by maintaining a check list on the status of everything in the home and making automation decisions based on the changing status of our system. However, our Android application can't directly access this data. The Raspberry pi must instead utilize the Apache webserver in order to update a MySQL database with the status of everything.

This MySQL database can then be referenced by the Android application through the Apache webserver in order to get the updated status of the home. This is necessary in order to keep the security and energy saving aspects of the home viable even when the Android application is not at home.

4.2.3 PHP

PHP is a programming language that is primarily utilized with servers over the internet. PHP can be added to our Apache webserver in order to give us a broader range of functionality to our webserver. PHP can be added to a webserver currently running Apache by following this guide: [10]
Once PHP has been configured on the webserver currently running Apache, the webserver can now process commands for database access and controlling our home system in real time. This can be done by utilizing an appropriate set of PHP scripts to give our Raspberry Pi and Android application access to the MySQL database and to allow our Android application to send commands to the Raspberry Pi.

10 http://www.thesitewizard.com/php/install-php-5-apache-windows.shtml

4.2.4 WebIOPI

This is the most straightforward way to provide web connectivity to our Raspberry Pi turned smart hub in order to connect it to our Android application. However, this is a pre-made firmware that works on a Raspberry Pi. The only customization would be to make the Raspberry Pi talk to our specific application and to open our router for the connection. We will choose this option provided we can not get an Apache webserver with PHP functional with our Android application.

4.2.5 Webserver Specifics

Our project relies on the Android application being able to communicate with our central hub. There are many options we could have chosen to bridge this communications, but we considered the future use of multiple Android applications concurrently and decided to go with an approach that would relay communications in a timely manner and utilize one synchronized database.

Currently our web-server is being run off of a laptop and being broadcast locally to the Android application and central hub. We have not implemented any form of encryption or other security measures since this is simply a prototype. We can move everything over to an actual http server and implement appropriate encryption once we leave the prototyping phase.

We decided to utilize MySQL for our database implementation. This allows our central hub and Android application(s) to perform tasks simultaneously without interruption or running the risk of becoming out of sync. Our database design is very simple. We are utilizing two different tables: one to keep track of our modules' status and one to relay commands from the Android application(s) to our central hub. Each module represents a row in the status table and the command table. This allows our central hub to keep track of everything individually and our Android application(s) to keep track of everything at once.

In order to cut down on the number of individual programs needed for server communications, we chose to utilize PHP to give access to our database to the Android application(s) and central hub. PHP allows us to have multiple live connections with our database and allows us to use the same set of programs for our Android application and central hub. This cut down drastically on our time for implementation and makes operations more efficient.

Our PHP programs are designed to give the ability to query and update the database to our Android application and central hub. There are 11 PHP programs designed for the status table and 15 PHP programs designed for the commands table. Adding more peripherals to our project will require the addition of more PHP programs. However, once the project leaves prototyping, many of the PHP programs will be combined in order to cut down on unnecessary server communications.

The 11 PHP programs for the status table are made up of six query programs and five update programs. The five update programs are mainly there to allow the central hub to change the statuses of the hardware peripherals. Five of the six query programs are mainly there to allow the Android application to check the status of any one of the hardware peripherals. The sixth query program is there to allow the central hub and Android application to check the status of all of the hardware peripherals simultaneously.

The 15 PHP programs for the commands table are made up of ten update programs and five query programs. Five of the update programs are there to allow the Android application to relay a command to the central hub by changing a status on the database. The five query programs are there for the central hub to be able to see if any of the statuses have been changed by the Android application. The other five update programs are there for the central hub to change the status back after performing the appropriate command.

The overall functionality of our project is as follows. Whenever the central hub detects a change in status for one of the hardware peripherals, it executes the appropriate PHP program to change the status listed in the database. This means that whenever the Android application executes one of the PHP programs to query one of the hardware peripherals status, that the status it is receiving is live. Whenever the Android application wants to change the status of one of the hardware peripherals, it executes the appropriate PHP program to change the status listed in the database. Whenever the central hub detects this change in status, it changes the appropriate design peripheral and then calls on the appropriate PHP program to change the status back.

5.1 Hub

The chosen developmental board that we decided upon for the hub of our Close to Home system was the Raspberry Pi Model B. This, paired with an Xbee Radio Module interfaced with the Raspberry Pi's GPIO headers allow for the hub of Close to Home to communicate with the various door locks, window locks, lights, room sensors, and electronic devices for efficient and timely monitoring and application. Within the following paragraphs, we will explain at length why we chose to utilize the Raspberry Pi as our hub, what it means to use the Raspberry Pi, and a lengthy explanation of all of the different aspects of the Raspberry Pi that made it a good fit for Close to Home.

There exist two distinct versions of the Raspberry Pi, one predating the other, which contain very different sets of hardware in some aspects, and many similarities in others. Given that we needed to choose the better of the two for our project, we will outline the two options below: (See Next Page)

	Model A	Model B
Target Price:	$25.00	$35.00
System on a Chip:	Broadcom BCM2835 (CPU, GPU, DSP, SDRAM, and single USB Port)	
CPU:	700 MHz ARM1176JZF-S core (ARM11 family, ARMv6 instruction set)	
GPU:	Broadcom VideoCore IV @ 250MHz OpenGL ES 2.0 (24 GFLOPS) MPEG-2 and VC-1 (with license), 1080p30 h.264/MPEG-4 AVC high-profile decoder and encoder	
Memory (SDRAM):	256MB (Shared with GPU)	512MB (Shared with GPU)
USB 2.0 Ports:	1 (direct from BCM2835 chip)	2 (via built-in integrated 3-port USB hub)
Onboard Storage:	SD / MMC / SDIO Card Slot (3.3V card power support only)	
Onboard Network	None	10/100 Ethernet (8P8C) USB Adapter on the third port of the USB hub.
Low-Level Peripherals:	8 x GPIO, UART, I^2C bus, SPI bus with two chip selects, I^2S audio, +3.3V, +5V, Ground.	
Power Ratings:	300 mA (1.5W)	700mA (3.5W)
Power Source:	5V via MicroUSB or GPIO Header	
Size:	85.60 x 53.98 mm (33.70 x 2.125 in)	
Weight:	45g (1.6oz)	
Operating System Support:	Arch Linux ARM, Fedora, FreeBSD, NetBSD, Debian GNU/Linux, Raspian OS, Plan 9, RISC OS, Slackware Linux	

Table 5.1: Printed with permission from Raspberry Pi Foundation

The Raspberry Pi is a credit-card-sized single-board computer that was initially developed by the Raspberry Pi Foundation in the UK. It was created with the purpose of teaching basic computer science in elementary schools worldwide by providing the teachers with an

affordable computer that has access to basic functions and low-level computation. On board the Raspberry Pi is a Broadcom BCM2835 system on a chip which has within it an ARM1176JZF-S 700Mhz processor. Within the built-in firmware, there also exists the “Turbo Mode” feature, which allows for a normally unprecedented within-warranty overclocking, allowing the Raspberry Pi to achieve speeds of up to 1Ghz without voiding any agreement with the manufacturer.

This processor was perfect for our group's needs, as we needed a developmental board containing a processor that didn't have too much power, in an effort to keep the cost down, but still has the capability to manage the Close to Home network with relative ease. Given the efficiency of ARM processors, and their use of the RISC (Reduced Instruction Set Computing) code, the modest 700Mhz of processing power was ample for our needs. In the event that the processor was becoming overwhelmed with the task at hand, it would have been rather simple to add a small heatsink to the top of the processor, and perform a bit of overclocking to bring the speed up to 1Ghz, but this was not exclusively necessary for the environment that we had set up. The alternative to the RISC architecture is the CISC (Complex Instruction Set Computing) architecture, which is what most modern computers use. We'll outline the differences below:

CISC	RISC
Complex instructions taking multiple cycles	Simple instructions taking 1 cycle
Any instruction may reference memory	Only LOADS/STORES reference memory
Not pipelined or less pipelined	Highly pipelined
Instructions interpreted by the microprogram	Instructions executed by the hardware
Variable format instructions	Fixed format instructions
Many instructions and modes	Few instructions and modes
Complexity in the microprogram	Complexity is in the compiler
Single register set	Multiple register sets

The choice of the processor was also a personal one for many of our group members, as development with ARM processors is becoming more and more prevalent in the job market these days. With the rise of smartphones and other mobile computing, any low power consumption and low instruction set processor is becoming the talk of the town

among Silicon Valley. Obviously, if something is becoming popular in modern technology, Computer and Electrical Engineers need to be at the forefront of the field.

Boasting about a previous Senior Design project that involved working with an ARM-based processor looks wonderful on a resume, and we hope that the experience we gain with this project will aid us in our search for work in related fields. Ultimately, we would love to get hired on at a company where we can bring Close to Home to many homes around the world, but that will remain a pipe dream for now, as we get a working and finished prototype online as a first priority.

Given that the software we develop might require the hub to keep a wealth of information loaded into memory, it was important to choose a board that had enough RAM for the task. The Close to Home system will involve the simultaneous interaction of many aspects of the house for monitoring and maintenance purposes, so this need for memory is a hard requirement, and necessary in order for the system to function properly. The Raspberry Pi has two versions, the earlier released Model A which comes with 256MB of RAM, and the later revised Model B which comes equipped with 512MB of RAM. Both of these units share their memory with the GPU system—which is of little consequence as we'll explain later—and the price points of the Model A and Model B are $25.00 and $35.00, respectively. Given that it is a mere $10.00 difference to nearly double the amount of available RAM we have, it was a fairly obvious choice. In the event that we had a finished system that performed very well, but didn't have enough RAM to accomplish the final prototype's task, it would have seemed foolish to skimp out on the $10.00 that we could have spent building a better foundation.

Not included in the above table is the various forms of video and audio output and input that the Raspberry Pi has access to. Largely, we will not be using the audio and display capabilities of the Raspberry Pi due to the fact that all interaction with the Close to Home system will be done on a user's smartphone, leaving the hub to silently and dutifully carry out its actions without the intervening eyes of others. We will likely be using some crude form of displays for debugging purposes as we get the Close to Home hub to a stable state, but following shortly thereafter, this code will be removed, and the hub will operate without any direct interaction whatsoever. We considered the fact that having superfluous features such as these on our developmental board may

prove to be a money sink, but considering the low cost of the Raspberry Pi, it was not a measurable loss by any means.

The Raspberry Pi's onboard “System on a Chip”: The Broadcom BCM2853, came fully equipped with a mini-UART system, which will prove invaluable in our project. Since we will be using the ZigBee trasmission protocol, we will likely need to follow the transmission of data every step of the way, from handshakes to FIFO to flow control and back. Listed online at Broadcom's website is a datasheet for the BCM2853, which wonderfully outlines the mini-UART's operation registers, and has complete instructions for setting the bits high and low for the desired steps of the transmission process. As well, it has a full list of the capabilities of the on-board mini-UART system, and what it is lacking, if anything. The mini Uart has the following features:

- 7 or 8 bit operation.
- 1 start and 1 stop bit.
- No parities.
- Break generation.
- 8 symbols deep FIFOs for receive and transmit.
- SW controlled RTS, SW readable CTS.
- Auto flow control with programmable FIFO level.
- 16550 like registers.
- Baudrate derived from system clock.

This is a mini UART and it does NOT have the following capabilities:

- Break detection
- Framing errors detection.
- Parity bit
- Receive Time-out interrupt
- DCD, DSR, DTR or RI signals.

The GPIO headers on-board the Raspberry Pi provided us with nearly unlimited choices as to what kind of external hardware we could add to the developmental board, in order to achieve the Close to Home hub that we envisioned. The layout of the analog and digital headers was intuitive and simple to understand, with information as to what each pin corresponded to readily available on the internet. What follows is a generalized version of the GPIO pinout for the Raspberry Pi Model B, as derived from online sources:

	Bottom	Top
Pin 1 // Pin 2	3.3V Power	5V Power
Pin 3 // Pin 4	GPIO 2 (SDA)	5V Power
Pin 5 // Pin 6	GPIO 3 (SCL)	Ground
Pin 7 // Pin 8	GPIO 4 (GPCLK0)	GPIO 14 (TXD)
Pin 9 // Pin 10	Ground	GPIO 15 (RXD)
Pin 11 // Pin 12	GPIO 17	GPIO 18 (PCM_CLK)
Pin 13 // Pin 14	GPIO 27	Ground
Pin 15 // Pin 16	GPIO 22	GPIO 23
Pin 17 // Pin 18	3V3 Power	GPIO 24
Pin 19 // Pin 20	GPIO 10 (MOSI)	Ground
Pin 21 // Pin 22	GPIO 9 (MISO)	GPIO 25
Pin 23 // Pin 24	GPIO 11 (SCLK)	GPIO 8 (CE0)
Pin 25 // Pin 26	Ground	GPIO 7 (CE1)

Table 5.2: Printed with permission from eLinux.org

What truly benefited the Close to Home project the most was having a developmental board that had as much online documentation as the Raspberry Pi. No other board that we were able to find online had as much extensive information and comparative application from other experiments as this board, and choosing it as the go-to device was an obvious, but very influential decision. The GPIO headers that tied in with the ARM v6 processor can be seen below:

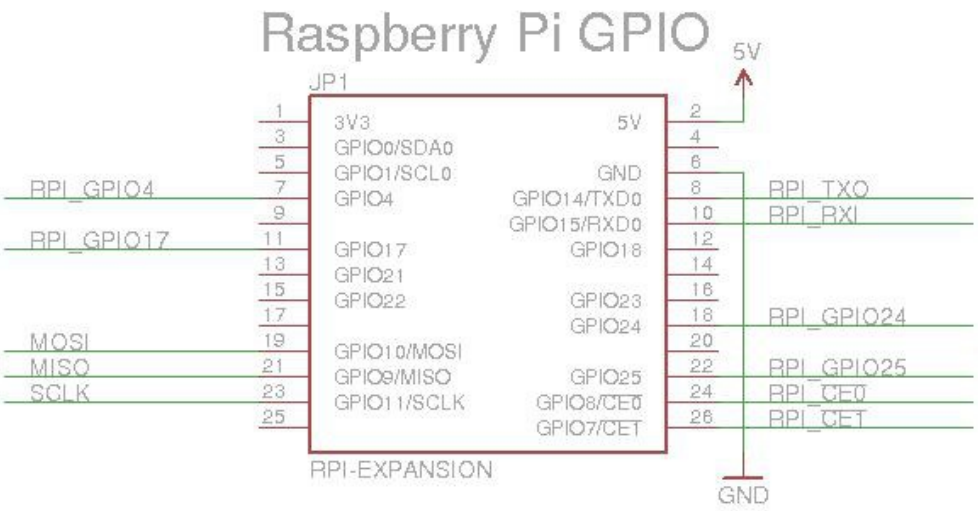


Fig 5.1: Printed with permission from bootc.net

On the above diagram, we plan to use GPIO 14 and 15 primarily with our Zigbee compatible extension to the board, allowing us to transfer easily over the low-power wireless interface. After that, we will need to simply make use of one of the ground connections, and provide 3.3v power to the unit via the GPIO ports, and we should be good to go.

The Raspberry Pi provided us with several outlets for use with any device that we ended up having to use. Despite the fact that the chosen modules and other devices changed several times before our final prototype, the Raspberry Pi's multiple GPIO headers provided a modular way for us to decide which item to use without having to deal with getting a piece of hardware that was specifically made for the device or not. This helped ease a lot of our minds as we were able to purchase only the things that we needed and none of the things that we didn't.

As you can see from the above diagram, and from the table provided earlier, the various Pinouts of the Raspberry Pi allow us to make use of many features simultaneously, and to interface with the board on a lower level of computation. By interfacing the GPIO headers directly with a Zigbee addon module via its I2C transmission mode, it was possible to provide our network with simple Zigbee communication without having to muddle around too much with alternatives. The process of attaching this module was one that we could have handled

in various other ways, as the Raspberry Pi is fully capable of Rx and Tx modes involving all aspects of a stack communication. For example, pins 8 and 10 are used for the transfer and receipt of single bit messages sent over the wire. Given that its possible to control the transmission of data at this low level, we would have been able to break apart the stack communication of the entire Zigbee packet, then send and receive the transmission of the necessary on or off information from the modules in a convenient and lightweight fashion.

On top of that, having access to pin 23 allowed us to have full access to the clock speed and cycle of the Raspberry Pi, ensuring that our data would always come into transmission at the proper time. It's important that we get data at the proper intervals when it comes to home monitoring and maintenance, else we might have had to deal with a situation wherein the time was off. If the time was off, we might have registered a device being turned off at the certain period of time that it needed to be on, and that would have a cascading effect on the rest of our modules as they attempt to work in unison. By having direct access and visibility of the cycle time of the Rasbperry Pi, we were able to create a system that ran smoothly and effectively at various intervals throughout the day.

Having access to the powerful database provided by the Raspberry Pi's quick cycle time and accurate testing facilities provided us with a way to verify that the entire Close to Home system was working properly. The packets sent by the various modules were able to be quickly heard by the Raspberry Pi, reacted to, and an acknowledgment signal could be sent in response. This signal was appropriate in order to keep the Close to Home system working in full tilt throughout the various day to day operation that a home user would be taking advantage of in the final product. For now, it was simple to have a looping program take advantage of the powerful Raspberry Pi's ARM1176JZF-S processor on a chip in order to cycle through its own program faster than the MSP430F1xx wall modules could cycle through theirs, letting the Raspberry Pi be the powerful central hub that it needed to be. The power requirement was necessary due to the fact that multiple modules throughout the home could be utlized at the same time, needing a system that was speedy, affordable, without having to dip too far into the parallel processing category of computation in order to avoid any runtime errors.

The chosen developmental board also had a slew of other helpful, but not altogether necessary features. For example, the HDMI out on the board allowed us to perform visual troubleshooting in a much more convenient fashion. We had a simple terminal version of Linux deployed for this purpose, and were able to fully browse a file structure and complete several testing steps by using the HDMI out port to a television that we had on-hand in the testing environment. The ease of use of this additional display out, and the fact that it did not cost us anything extra, since both versions of the Raspberry Pi had this feature, made it something extremely beneficial to the project as a whole.

The BCM2835 contained on the Raspberry Pi is also a powerful system in and of itself, boasting a blazing 700 Mhz transmission of data along the South Bridge and North Bridge simultaneously. The Raspberry Pi's ARM1176JZF-S processor is able to keep up with the various modules throughout the system by using the Slice of Pi's Xbee expansion board to communicate over the serial interface, while simultaneously communicating with the database via URL-executed PHP scripts. By merely opening the URLs as necessary, it was relatively simple to change the status of the devices, recording them as necessary and cataloging them for later use by the Android device.

The Raspberry Pi also boasted a great power savings benefit, yielding altogether a 40% power gain over the household compared to a household that ran the Raspberry Pi's measly 12v phone charger. It was basically akin to having a small smart phone taking power from a house to charge its battery, but only on a 10% scale. The benefits of the Close to Home System really outweigh the cons, as can be plainly seen by looking at the system in action.

Delving deeper into the functions built into the Broadcom BCM2835 System on a Chip that is embedded in the Raspberry Pi, we see so much more potential than offered by other boards. The documentation for this System is extensive, allowing us full information on all of the associated registers and memory access available on the chip. As a preview, I'll provide the following Table of Registry Maps for some of the lower-level functions on the BCM2835:

Auxiliary peripherals Register Map (offset = 0x7E21 5000)			
Address	Register Name	Description	Size
0x7E21 5000	AUX_IRQ	Auxiliary Interrupt Status	3
0x7E21 5004	AUX_ENABLES	Auxiliary Enables	3
0x7E21 5040	AUX_MU_IO_REG	Mini UART I/O Data	8
0x7E21 5044	AUX_MU_IER_REG	Mini UART Interrupt Enable	8
0x7E21 5048	AUX_MU_IIR_REG	Mini UART Interrupt Identify	8
0x7E21 504C	AUX_MU_LCR_REG	Mini UART Line Control	8
0x7E21 5050	AUX_MU_MCR_REG	Mini UART Modem Control	8
0x7E21 5054	AUX_MU_LSR_REG	Mini UART Line Status	8
0x7E21 5058	AUX_MU_MSR_REG	Mini UART Modem Status	8
0x7E21 505C	AUX_MU_SCRATCH	Mini UART Scratch	8
0x7E21 5060	AUX_MU_CNTL_REG	Mini UART Extra Control	8
0x7E21 5064	AUX_MU_STAT_REG	Mini UART Extra Status	32
0x7E21 5068	AUX_MU_BAUD_REG	Mini UART Baudrate	16

Table 5.3: Printed with permission from eLinux.org

In the above table, you can find several registers devoted to the Mini UART functions on-board the BCM2835, and having direct access to these registers will prove invaluable as we finalize a system that is able to process and receive data from the various satellite modules installed throughout the home. This kind of direct interaction with registers may prove to be unnecessary, as we will likely have firmware to help support our endeavors to this end, but having the option is certainly something that we can be thankful for, and something that we can use as leverage against the other competition in our market. If we are able to make communication between the hub and the satellite modules as error-free and quick as possible, than we can provide an unparalleled amount of service to our customer; something that we truly need to be providing as a number one goal.

What is listed above just scratches the surface of the amount of register information and supported documentation there is on the Raspberry Pi, and providing the reader with the exhaustive list in this document would be superfluous. It is enough to know that we have chosen a board with a highly supported processor, and we have full access to the

capabilities of the full system on a chip without having to pigeon-hole ourselves only to known hardware. This is important, as we can make Close to Home as large or as small as necessary, given the ability for our system to grow or shrink at will. This will provide our users with an expanding or shrinking budget for their needs, as the situation warrants.

Given that we will likely be using the Raspberry Pi's built-in SD Card loader and booter for our source software, we will be required to know every step of the boot procedure when an SD Card with data is interfaced with the board. What follows is a schematic of the SD Card interfacing with the BCM2835 System on a Chip, with some parts omitted for space restraints:

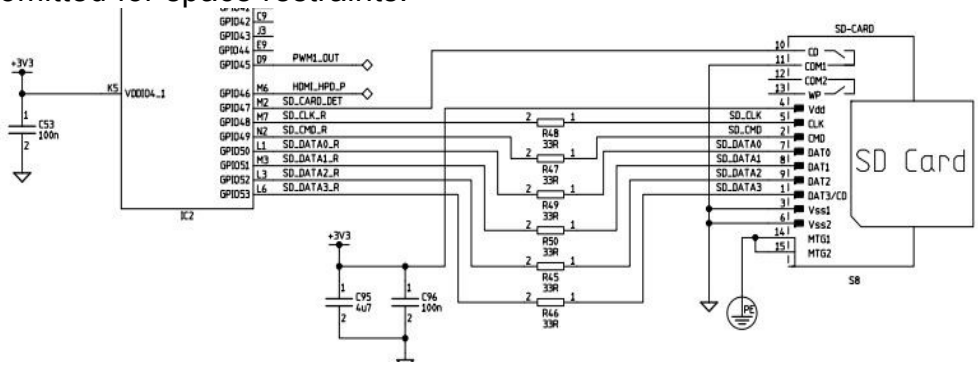


Figure 5.2: Printed with permission from RaspberryPi.org

In the above diagram, through the rather small print font, you can see that the stack of SD.CLK, SD.CMD, and SD.DATA0-3 are used during any interfacing with the SD Card module, and it should turn out rather simple to either write code that is compatible with this kind of interface, or at the very least find some manner of Linux distro to make use of the SD Card's boot-on-wake functionality.

What follows is an in-depth figure of the Raspberry Pi's built-in schematic. Therein you can see the power inputs, as well as the register operands of the Raspberry Pi's System on a Chip, the Broadcom BCM2835:

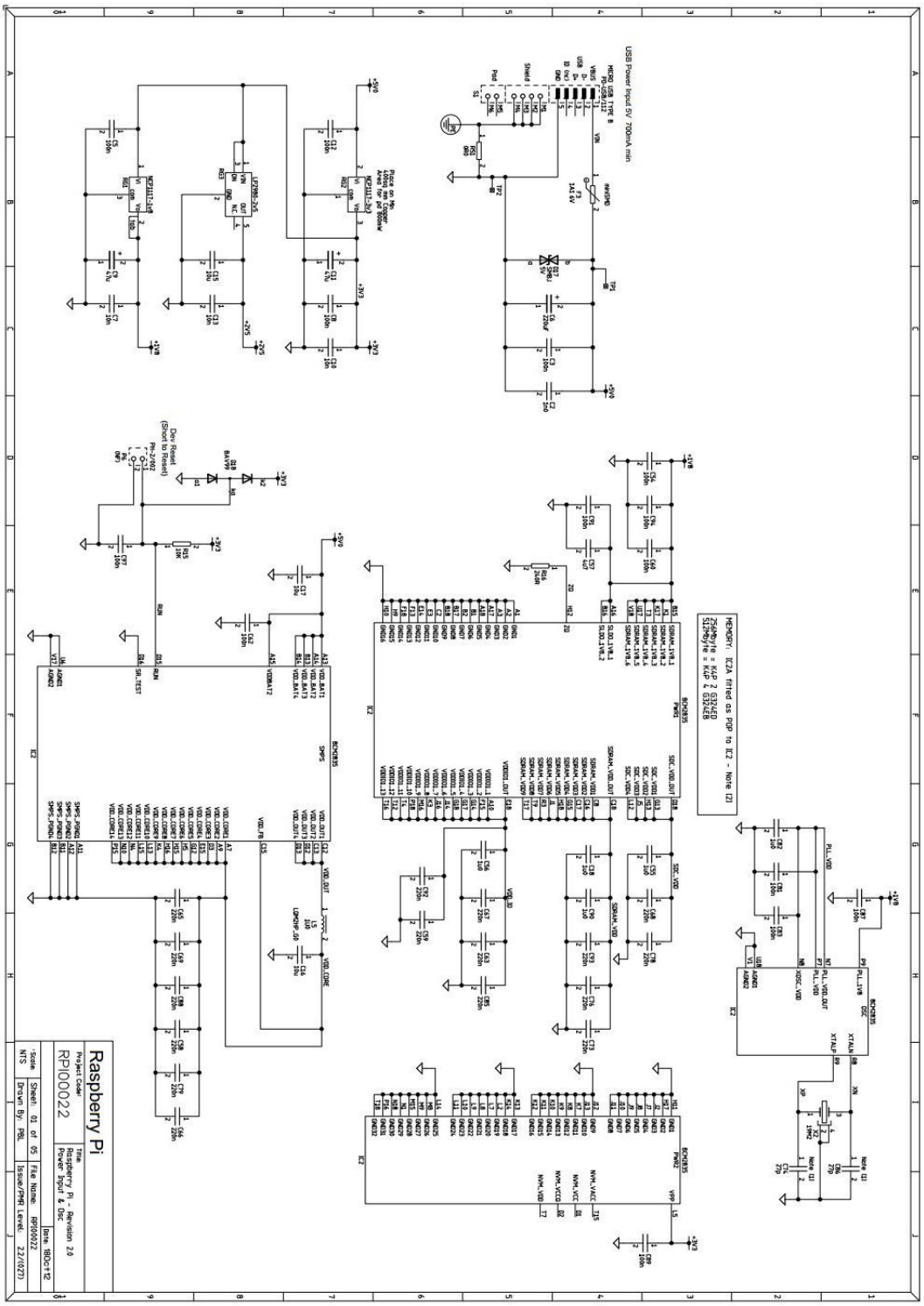


Figure 5.3: Raspberry Pi Schematic

Printed with permission from Raspberrypi.org

5.2 C2H Modules

In this section, we will be talking about the technical aspects of the circuitry. Please refer to section 3.4 for the reasoning and research behind these selections. Our final parts list is as follows:

Part	Part Number
Micro Controller	MSP430F2121
Resistor	-
Capacitor	-
Op Amp	TL084
Wireless Module	CC3000
Relay	Philmore 86-103

Table 5.1: Part List for In-Wall Module

Part	Part Number
Micro Controller	MSP430F2121
Resistor	-
Capacitor	-
Op Amp	TL084
Wireless Module	CC3000
Relay	Philmore 86-103

Table 5.2: Part List for 120V Module

Part	Part Number
Micro Controller	MSP430F2121
Resistor	-
Capacitor	-
Op Amp	TL084
Wireless Module	CC3000
Relay	

Table 5.3: Part List for 240V Module

Part	Part Number
Micro Controller	MSP430F2121
Resistor	-
Capacitor	-
Op Amp	TL084
Micro Switch	S3264
Wireless Module	CC3000
Relay	Philmore 86-103
USB Cable	Standard

Table 5.4: Part List for Computer Module

Part	Part Number
Micro Controller	MSP430F2121
Resistor	-
Capacitor	-
Wireless Module	CC3000
Servo	FUTM0303

Table 5.5: Part List for Lock Module

5.3 Micro Controller

The micro controller we have chosen for all of the modules is the MSP430F2121. We explain the process of elimination that leads us to this decision in section 3.4. The following are the parametrics on the chip we have selected.

Voltage Range	1.8V to 3.6V
Power Consumption	
Active:	250 uA
Standby	.7 uA
Off (RAM Retention)	.1 uA
Memory:	
Flash	8KB
RAM	256KB

Table 5.6: MSP430F2121 Parametrics (Printed with permission from Atmel Co.)

The main draw of this board is its especially low standby operating value. This allows us to not worry about the draw this board will have if it is in one of the modules that is powered off of the wall outlet, and it makes it possible to power the board on the batteries we are hoping to run the modules on (Section 5.6).

5.3.1 Microcontroller Programming

The MSP430 can be optimally programmed in two different languages. These two languages are C and Assembly. Since we are running some fairly complex commands through the Zigebee stack and the like, we have chosen to use C as the language for the microcontroller. The programming for the microcontroller will not be as complex as the programming for the hub, app, or webserver will be, as microcontrollers are generally worried about electrical signals coming from the components wired into the microcontrollers. With this said, the majority of the programming will be simple IF and CASE statements that set finite state machines. For example, for the lock module, we are simply interested in setting our register for lock to '1' when the micro switch is not depressed, as this means that the bolt will be slid forward into the door, locking it. Aside from these statements running in the body of the code, we will be calling several procedures in order to push data to where it needs to be. These procedures include but are not limited to dealing with encryption, wireless communication, and general state checking. Each module will have its own programming, as each module has a unique set of inputs and outputs, however the logic flow remains the same.

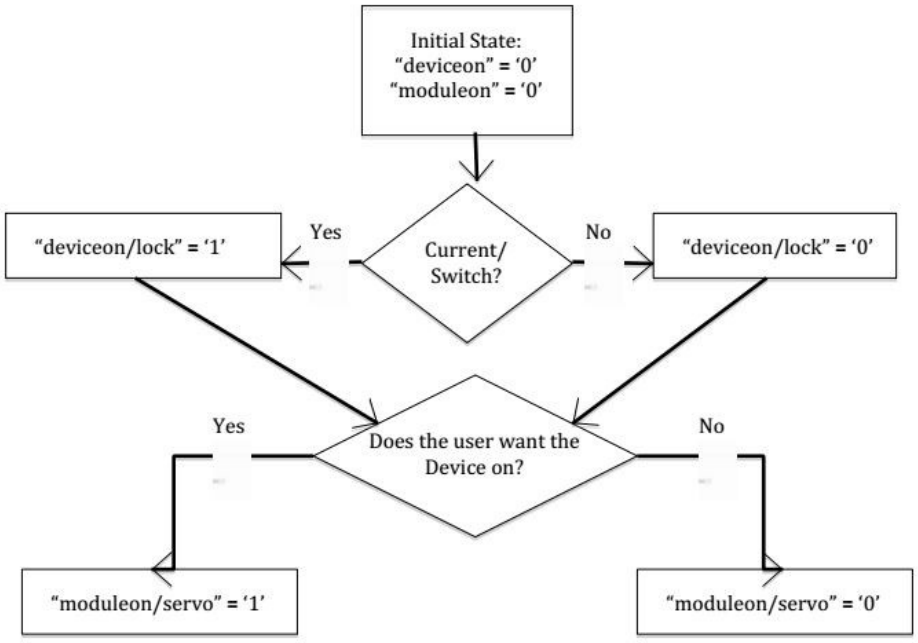


Figure 5.3.1: Flow Chart for Microcontroller Programming

From the flow chart, we can see that the code for the microcontroller is simply going to be a series of state checking. To begin, every modules initial state will have both the deviceplugged into the module, and the module itself turned off. For the 120V/240V Modules, the USB Module and the In-Wall Module, the program will run to see if there is an increased current running through the module. If there is, the system will set the "deviceon" value to '1', otherwise it will be set to '0'. For the lock module, the program will check to see if the micro switch is depressed or not and set the "lock" status accordingly. Next, we will check to see whether the user wants the device to be on or not. This is a more complicated statement than it seems on the surface. We are essentially looking for two values here. First, has the user manually told the system that it wants this device on? If it has, this overwrites any other rules that have been written into the app or hub, and turn the device on or off, depending on the desire of the user. If the user has not manually adjusted the status of the device, the system will look to see if the device *should* be on based on the user-predefined settings, that is, is this device one when the room is vacant? How long has the room been vacant? Is this equal to or greater than the time the user has defined to turn appliance off?

These questions will be answered by calling the occupancy sensor's procedure. The occupancy sensor's microchip will be programmed in a simple manner. We will have the user step out of the room he or she is currently trying to program into the system, and we will have the value for "occupancy" = '0'. We will also have a timer, which will set itself to '0' as the system initializes and during the initial sync. Once the system has been initialized, the counter will begin counting. The counter will increment until the laser beam is broken, signifying that someone has passed in front of it. If the outside beam is broken first, followed by the inside beam, the system will assume someone has entered the room, at which time the value for "occupancy" will be set to '1'. If this occurs again, the value of "occupancy" will be incremented. It is important to keep track of how many people are in the room, as we do not want to shut the lights off if someone is still there by themselves. Once this value has been set to at least one, we will call a procedure that will check the user's settings. Does this user want certain appliances turned on when someone enters this room? The microcontroller will act accordingly. There is no point for a counter to be counting at this point, as there are no obvious advantages to knowing how long the room has been occupied, as it is not like a user would program the system to shut lights off while they are in the room. In the even that the user is in the room and wants appliances shut off, for example if they are going to sleep, or just want to turn an appliance off from across the room, the programming will not hinder this. The program will continue to run in this state until it sense the laser beam in the occupancy sensor being broken again. If the inside beam is broken first followed by the outside beam, the system will assume that the user has left the room. Similar to the procedure of multiple people ocuupying the room, if this event occurs again, it will decrement the value of "occupancy" until the value is at '0'. Once the value of "occupancy" has reached '0', the timer will start back up and begin to count up. The system must then call a procedure to pull the user's desired setting onto the local chip. The important settings being whether or not the user wants the system to automatically shut appliance off when the room is vacant, and which appliance the user wants shut off. After this information has been parsed, with every tick of the counter, the system will check and see if the desired number has been reached or not. Once the appropriate idle time has been reached, the system will send the proper command to the hub, which will in turn relay the proper status adjustment to the appropriate module.

5.4 Lock Status and Amp Meter

As mentioned in Section 3.4, for the each module has a sensor that we have to use to monitor the status of each device. For the Lock Module, we are using a micro switch. We have selected a micro switch with a roller on it, as the lever will be situated parallel to the bolt of the door. When the door is unlocked, the level will be depressed, when it is locked the lever will be up. We selected a switch with moderately low activation weight, and which was rated for as many lifetime cycles as we could find. The following are the parametrics for the S3264 Micro Switch:

Contact Rating	5A @ 250V
Mechanical Life	5,000,000 Cycles
Electrical Life	500,000 Cycles

Table 5.7: S3264 Parametrics. (Printed with permission from Altronics Inc)

The other modules will not be using a micro switch for status detection, but rather an amp meter to measure current. The micro controller will be programmed to recognize the increase in current, and set the device status to “on”. Unfortunately, there is not direct way to measure the passing current using the micro controller, however we can use the Analog to Digital Converter (ADC) on the MSP430F2121 to measure a voltage drop across a resistor. We are still doing research for the exact number of the resistor value.

5.5 Servo

The servo we have chosen for the project is the FUTM0303. The FUTM0303 is a high-torque miniature servo, that takes up little space, and provides more than enough torque for what we need. The following is the parametrics for the FUTM0303:

Power Supply	4.8V
Torque	51 oz-in @ 4.8V
Dimensions	1.2” x 0.4” x 1.1"

Table 5.8: FUTM0303

We are planning on using 9 Volts for the power supply for the door, which will provide enough voltage to run it at 51 oz-in. There is a rating for 6 Volts, however the manufacturer recommends not using this setting based on wear and tear concerns. The dimensions for this servo are great, as each parameter is fairly small, and will inside the door itself, as an average door is about 2” thick.

5.6 Kill Switch (Relay)

For the modules that cut the power to the device to turn it off, we need a relay to turn the power on or off depending on the desired state of the device. For the modules that will be using a standard 120V outlet as its power source, we will be using the Philmore 86-103. This is a 120V Single Pole Double Throw (SPDT) relay that can handle the voltage load from a wall, and that can be switched using the current we are able to produce. We also picked this relay because it is PCB mountable which will be extremely useful during testing.

5.7 Power Supply

Choosing the proper power supply was vital for the project's success. Given that the entire purpose of the Close to Home system—and most importantly, our funding—depends solely on creating something that saves our users on their monthly power bills, this can be heavily influenced by the choice of power supply. Below we will list the final choices we've made for providing power to our units, and the reasons for why we chose them with detailed tables and charts.

5.7.1 In-Wall Power

As mentioned previously, the In-Wall Module, 120V Module, and 240V Module will be run from the wall source. Obviously we will not be running the full 120V or 240V through the board, as the MSP430 runs on the range between 1.8V and 3.6. We will build a circuit to step this down and convert it to DC so that the board can be powered properly.

One serious issue with running anything from a wall outlet is the risk of power outages. If no power is run to the board, the RAM can very easily

lose all of its data as RAM is volatile memory. So, for each board, in the event of power failure, we will install a small watch cell battery capable of putting out the current needed for RAM Retention.

An industry standard for things like this is the CR2302 3v Lithium Watch Cell. After doing some research we found that while the CR2302 is a standard size, the capacities can vary depending on the company that manufactured it. We are of course more interested in the highest capacity, so we have chosen the Energizer brand battery for our boards. A comparison of each brand can be found in the following table:

Brand	Energizer	Sanyo	Maxwell	Panasonic
Capacity(mAh)	240	200	220	210

The Energizer brand boasts an impressive capacity of 240 mAh. The math is simple for determining the battery life:

This means, that with a battery with a capacity of 240 mAh, we can run a board's RAM for 1,680,000 hours, or roughly 192 years. This is a lot longer than the actual estimated life of the battery itself, or any other component.

5.7.2 Battery Power

Since the lock modules will be installed in the door itself, and not attach to a wall power outlet like the other modules, we need a self-contained power source for these. We have to take into account everything that we are powering on the board. The only two sources that will need to be actively powered are the MSP430 itself, and the Servo.

The MSP430 is rated to run between 1.8V to 3.6V, and the servo runs at 4.8V. With these two values in mind, the 9V battery supply should work for us.

5.10 Circuitry Housing

As with any project that involves circuitry, we will need to properly house everything to make sure there are no shorts or outside disturbance to the circuitry. While there are many ways to build this housing, as mentioned in the Research section, we have chosen to use

3D Printing for our housing. We will be using a RoBo 3D Printer, which is owned and operated by a friend of the group. The RoBo 3d Printer is a PLA printer and has the available dimensions for the type of housing we need. As mentioned in the research section, we will be able to use a wide variety of colors, but we will be using black PLA for this, as it is readily available, and a neutral color. Obviously for prototyping purposes we are not looking for everything to be cosmetically pleasing, but it's important to know that we have an option to make it that way in case we do decide to replicate this process for mass production. Even though we have six different types of modules, we only need to create three unique housings. The three types we will be making are for the plug modules, the USB module, and the occupancy module. The lock module itself would need housing, however we are planning to situate the board, servo and micro switch inside an already fabricated lock mechanism for a door.

For the first module, we are designing the housing for the plug module. We have already established that our PCB will be approximately 50mm by 65mm. For clearance, we will be printing the housing to be 55mm by 70mm. We want a very sleek, minimal design because we would like our modules to be evasive. To accomplish this, we want to design a simple rectangular box, with a female plug on one side, and a male plug on the opposite side. We have rendered a 3D model of what we would like the finished prototype to look like.

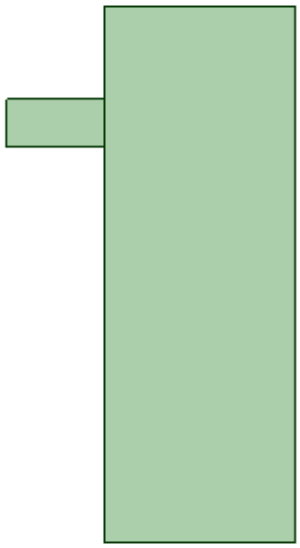


Figure 5.10.1: Side View of Plug Module

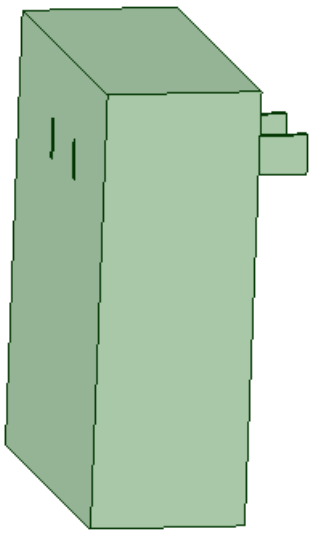


Figure 5.10.2: 3D View of Plug Module

As you can see from the renderings, we are looking to keep these as simple as possible. Essentially if no one were the wiser, it would look like a standard power brick. The overall dimensions of the finished prototype will be 55mm x 77mm x 14mm just the box and 55mm x

77mm x 26mm to the tip of the power prongs. We chose the depth for a couple of reasons. The first reason is that standard power prongs are 12 mm long. Since we will need some clearance for the input prongs along with the board width, we set our depth to be 14 mm. With this information, we of course have a minimum set. We look at the datasheets for all of our components and the next largest component is the relay. The relay is also 12mm from the board, so we know that a clearance of 14mm will work.

The next module we will design is the module for the occupancy sensor. The occupancy sensor consists of two separate modules. The first is the housing for the board and the laser module. Since this is a battery powered module, the batteries are going to be the main restriction on the size. According to the data sheet the battery is 44.5mm tall and 9mm in diameter. With four batteries side-by-side, this put the total battery space occupied is 44.5mm by 36mm, give or take a couple of mm for a small space between each battery. The depth must be the 9mm to allow for the battery, plus an additional clearance for the board and the laser diode. According to the data sheet, the diode is 8.5mm tall. This brings us to a total of about 18mm that we must make clearance for. We will make the box 52mm x 40mm x 20mm. The following is the rendering of the occupancy transmitter box:

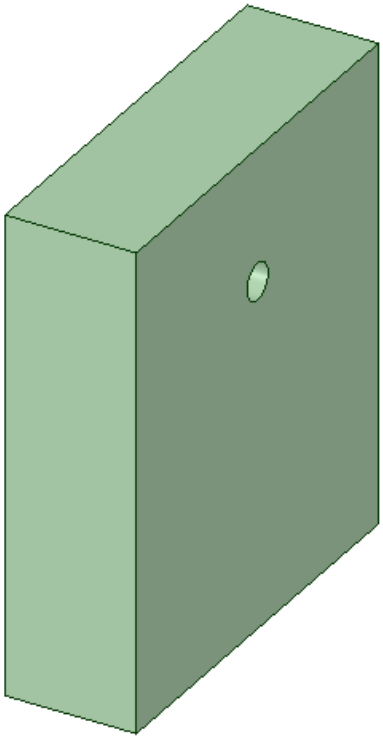


Figure 5.10.3: 3D View of Laser Transmitter

Again, we are looking to keep things as simple as possible. The occupancy sensors will essentially be out in the open, as they will be mounted to the doorframe. We want to keep them as inconspicuous as possible. In order to keep with this idea, the receiver housing must be even less conspicuous. While a black box may be hard to hide because of its dimensions, if we make the receiver housing small and round, we can actually set the receiver into a recess in the door frame. The following is an idea of our what the receiver will look like:

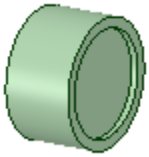


Figure 5.10.4: 3D View of Receiver

According to the datasheet, the diameter of the photocell is 5mm. As you can see from the rendering there is a recessed area. This inner circle is the 5mm of the photocell. The outer ring extends out to 6mm. This will keep the receiver module small, easy to hide, and give the receiver some padding just incase it receives physical contact from any outside source. The depth of the photocell according to the datasheet is 2.4mm, so we will make the housing 3mm deep to accommodate it.

Finally, our last module we will be printing housing for is the USB module. We had originally wanted to make this module small like a flash drive. Unfortunately within the scope of this project, it is not something we will be able to accomplish. Since we cannot do this, the module will consist of a box holding the wireless module and MSP430, and will connect via USB cable. While this is a simple idea, we felt the need to create a rendering for scale purposes, and to distinguish from or original idea of a thumb drive type form factor.

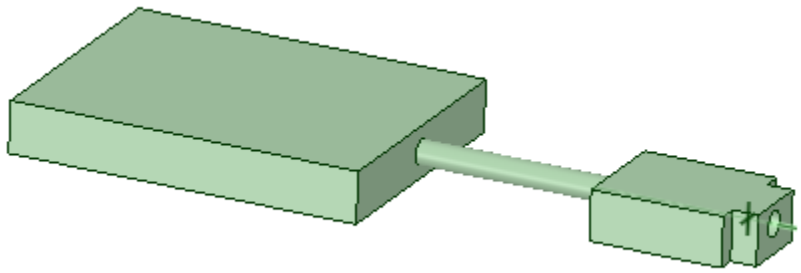


Figure 5.10.5: 3D View of USB Housing

The dimensions of this module are limited only by the size of the board on the left side, and limited by the physical space taken up by the USB dongle soldered into the wires from the cable. The board on the USB module will be much smaller than any of the other modules, as the components on the USB board are limited to the wireless antenna, the backup battery, and the MSP430. The USB is unique among the modules because the source of power is not on the board itself, but is powered through the USB cable. We are estimating the entire board taking up an area of 30mm x 50mm. This will make the module somewhat inconspicuous, as it will not take up any more space on a desk than a mouse or an external hard drive.

7.0 Prototype Testing

This section is dedicated to describing what needs to be tested, who will specifically be responsible for testing what, and how we will test it. Some of the parts will require very simple testing while some of the parts will require very extensive testing. The entire project as a whole will require a very rigorous routine of tests in order to make sure multiple users can use the system concurrently. This will require multiple people testing the android application while someone monitors the smart hub and/or the web server. The test scenarios described below will go through details on testing each part and the system as a whole. This process will allow us to verify that none of the individual parts are malfunctioning. Once we verify the individual parts are working, we can debug the system as a whole. Only then will we have a good idea as to how the entire project works together as a whole.

7.1 Component/System Testing Specifications

This section is dedicated to listing each system or component that needs to be tested and what needs to be tested about it. This will allow for a thorough testing of all individual components and then subsequently the systems of combined components. These tests will guarantee that any errors in assembling the prototype system are from our software control side and not the physical components. After all of the errors from combining the components have been debugged, we can begin testing specifics of the system as a whole.

System	Component	Testing Specifics
Doors	Door open/closed sensor	-Check accurate status of open/closed -Check ability to report status to the smart hub
Complete System	Door open/closed sensor	-Check the time and accuracy of reporting the status to the Android application
Windows	Window open/closed sensor	-Check accurate status of open/closed -Check ability to report status to the smart hub
Complete System	Window open/closed sensor	-Check the time and accuracy of reporting the status to the Android application
Locks	Locked/Unlocked control	-Check accurate status of locked/unlocked -Check ability to lock/unlock -Check ability to report status to the smart hub -Check the ability to receive commands from the smart hub
Complete System	Locked/Unlocked control	-Check the time and accuracy of reporting the status to the Android application -Check the ability of the Android application to manipulate the locked/unlocked control
Lights	on/off control	-Check accurate status of on/off -Check ability to turn on/off -Check ability to report status to the smart hub -Check the ability to receive commands from the smart hub

Complete System	Lights on/off control	-Check the time and accuracy of reporting the status to the Android application -Check the ability of the Android application to manipulate the on/off control
Fans	on/off control	-Check accurate status of on/off -Check ability to turn on/off -Check ability to report status to the smart hub -Check the ability to receive commands from the smart hub
Complete System	Fans on/off control	-Check the time and accuracy of reporting the status to the Android application -Check the ability of the Android application to manipulate the on/off control
Doorways	Laser Sensors	-Check accurate status of broken/unbroken -Check ability to work in tandem to determine the direction of travel -Check ability to report status to the smart hub
Complete System	Doorway Laser Sensors	-Check the time and accuracy of reporting the status to the smart hub by checking the timing for the smart hub to mark a room as active or inactive -Check the ability of the smart hub to register multiple people going through one set of doorway sensors by checking the smart hubs accuracy with total room count -Check the time and accuracy of reporting a change in room count to the

		Android application
Washer	on/off sensor	-Check accurate status of on/off -Check ability to report status to the smart hub
Complete System	Washer on/off sensor	-Check the time and accuracy of reporting the status to the Android application
Dryer	on/off sensor	-Check accurate status of on/off -Check ability to report status to the smart hub
Complete System	Dryer on/off sensor	-Check the time and accuracy of reporting the status to the Android application
Rooms	Temperature Sensor	-Check accurate status temperature -Check ability to report status to the smart hub
Complete System	Room's Temperature Sensor	-Check the time and accuracy of reporting the status to the Android application -Check the smart hubs usage of the individual temperatures in creating an average temperature
Rooms	Outlet Controls	-Check accurate status of on/off -Check ability to report status to the smart hub -Check the ability to receive commands from the smart hub
Complete System	Rooms Outlet Controls	-Check the ability for controlling the rooms power based on a room being marked active or inactive -Check the time and accuracy of reporting the status to the Android application
Rooms	Power Usage Sensor	-Check the ability to monitor the power of the outlet -Check the ability to relay the usage

		information to the smart hub
Complete System	Rooms Power Usage Sensor	-Check the ability to create an accurate picture of power consumption within the home -Check the time and accuracy of reporting the status to the Android application
A/c	Smart Thermostat	-Check the ability to regulate the temperatures -Check the ability to relay information to the smart hub -Check the ability to receive commands from the smart hub
Complete System	Smart Thermostat	-Check that the smart thermostat accurately changes the temperature based on the needs of the house -Check the time and accuracy of reporting the status to the Android application -Check the ability to receive commands from the Android application
Kitchen Appliance	on/off control	-Check accurate status of on/off -Check ability to report status to the smart hub -Check the ability to receive commands from the smart hub
Complete System	Kitchen Appliances on/off control	-Check the ability for controlling the appliances power from the Android application -Check the time and accuracy of reporting the status to the Android application
Smart Hub	Raspberry Pi	-Check the Raspberry Pi works

		-Check all of the intended firmware works -Check that the Raspberry Pi can communicate with all of the hardware peripherals -Check that the Raspberry Pi can issue commands to all of the hardware peripherals -Check that the Raspberry Pi can run our home automation algorithms -Check that the Raspberry Pi can interact with the Android application
Complete System	Smart Hub – Raspberry Pi	-Check that the smart hub can monitor the status of all of the hardware at the same time while also being able to issue controls to all of the appropriate hardware -Check that the smart hub accurately changes the status of rooms from active to inactive -Check that the smart hub manipulates the a/c based on the average temperatures and activity -Check that the smart hub effectively relays everything to the Android applications -Check that the smart hub accurately locks the doors when no one is home and powers down rooms when no one is in them
Android Applicatin	Android Application	-Check that the Android application can be used to control each of the hardware components -Check that the Android application can be used to check the status of each of the hardware components
Web Server	MySQL	-Check that the MySQL database is accurately updated with the status of the hardware components

Web Server	Apache	-Check that the Apache web server accurately relays data from the smart hub to the MySQL database -Check that the Apache web server accurately relays data from the MySQL database to the Android application
Complete System	Android Application, MySQL, Apache	-Check that multiple Android applications can concurrently check the status of the home and/or execute commands -It is known that executing contradictory commands at the same time will result in an error, so only executing different commands simultaneously should be tested

7.2 Who's Who of Testing

The testing will be done as a group. This will be done to eliminate the possibility for error and to ensure that one person is not responsible for a bulk of the testing. The testing of individual components will be divided up evenly. There will also be someone assigned to check the results of each testing procedure. This is to ensure that valid test results are admitted in order to guarantee that each individual component works. Otherwise when we combine all of the individual components, we would not know what the error was. This method of testing allows us to guarantee that any errors in combining the individual components are not from faulty components, but instead from our faulty arrangement of the components.

System	Component	Who's Testing	Who's Checking
Doors	open/close sensor	N. Godfrey	D. Krummen
Locks	locked/unlocked control	N. Godfrey	D. Krummen
Lights	on/off control	N. Godfrey	D. Krummen
Fans	on/off control	N. Godfrey	D. Krummen
Rooms	outlet controls	D. Krummen	N. Godfrey
Rooms	Power usage sensor	D. Krummen	N. Godfrey
Smart Hub	Raspberry Pi	D. Krummen	N. Godfrey
Android Application	Android Application	M. Garcia	J. Early
Web Server	Apache Web Server	J. Early	M. Garcia
Complete System	Everything (Details Below)	Everyone (Details Below)	Everyone (Details Below)

The testing of the complete system will require a more comprehensive, rigorous test routine that each of us must participate in. At this stage all of our individual efforts will be combined into one complete system. We will each be responsible for testing the aspect of the project that we designed, but it will involve all of us testing our various portions at the same time. This will allow us to guarantee that the system works flawlessly as a whole and does not break under any use cases. Below are the details of our comprehensive testing plans for the complete system.

The testing of the complete system will entail: checking the accuracy of hardware status on the smart hub and the Android application, checking the ability to issue a command to any of the hardware components from the Android Application, checking the ability for the smart hub to automate itself, and checking the ability of the system to handle concurrent users.

Feature	Who is testing it
B. Godfrey	Checking the accuracy of hardware status on the smart hub
D. Krummen	Checking the ability of the smart hub to automate itself
M. Garcia	Checking the ability of the Android application to command the hardware
J. Early	Checking the ability of the Android application to command the hardware

The final test for our complete system will entail testing its ability to handle multiple concurrent users. We will accomplish this by working as a group in order to verify the system works accurately and in a timely manner. Once we have completed this level of testing, the complete system will be confirmed as fully functional. We have listed the details for who will perform what below.

Bailey Godfrey and Marc Garcia will be tasked with going through all of the use case scenarios for the Android application separately on different instances of the Android application. They will also be responsible for setting off multiple sensors. This will guarantee that the system can handle multiple users checking hardware statuses and issuing commands at once while verifying that the home automation aspects work too. In order to verify that everything happens in a timely manner, Joshua Early will be tasked with monitoring the MySQL server and the Apache web server traffic. Checking the web connected elements will allow us to verify that the Android applications are in fact receiving real time status updates and executing their commands in real time. Daniel Krummen will be responsible for monitoring activity on the smart hub. Any activity on the smart hub must represent movements or commands made by Bailey or Marc and must be passed on to the web connected elements, MySQL and the Apache web server.

7.3 Testing Procedures

Here we have detailed how all of the components and their appropriate use case scenarios will be tested. The tests for the components as apart of a complete system are also detailed below. Following the table is our specific plan for testing the entire system by verifying that it can handle multiple concurrent users in real time.

System	Component	How it is being tested
Doors	open/close sensor	We will verify that the sensor powers on. Once we have verified the component works, we will verify that we can connect it to our smart hub by syncing the two devices through the appropriate firmware. Then we will open and close the door and see what response we get in order to verify that it responds to the door being open or closed and sends a unique value for both situations.
Windows	open/close sensor	We will verify that the sensor powers on. Once we have verified the component works, we will verify that we can connect it to our smart hub by syncing the two devices through the appropriate firmware. Then we will open and close the window and see what response we get in order to verify that it responds to the door being open or closed and sends a unique value for both situations.
Locks	Locked/ unlocked control	We will verify that the sensor powers on. Once we have verified the component works, we will verify that we can connect it to our smart hub by syncing the two devices through the appropriate firmware. Then we will open and close the window and see what response we get in order to verify that it responds to

		the door being open or closed and sends a unique value for both situations.
Lights	on/off control	We will verify that the controller powers on. Once we have verified the component works, we will verify that we can connect it to our smart hub by syncing the two devices through the appropriate firmware. We will then install the control into its respective light. Then we will power the light on and off and verify that a unique value is sent to the smart hub for each situation. Then we will verify that the smart hub can turn the light on and off.
Fans	on/off control	We will verify that the controller powers on. Once we have verified the component works, we will verify that we can connect it to our smart hub by syncing the two devices through the appropriate firmware. We will then install the control into its respective fan. Then we will power the fan on and off and verify that a unique value is sent to the smart hub for each situation. Then we will verify that the smart hub can turn the fan on and off.
Doorways	laser sensors	We will verify that the sensor powers on. Once we have verified the component works, we will verify that we can connect it to our smart hub by syncing the two devices through the appropriate firmware. Then we will verify that the

		sensors send unique values for broken/unbroken. We will then install the sensors into their respective doorway in the appropriate fashion. After that we will verify that both sensors send their signals back to the smart hub at different times, indicating which sensor the user passed through first.
Washer	on/off sensor	We will verify that the sensor powers on. Once we have verified the component works, we will verify that we can connect it to our smart hub by syncing the two devices through the appropriate firmware. Then we will install the sensor on the back of the washer. Once we have the sensor installed, we will turn the washer on to verify that a unique value is sent to the smart hub for off and on.
Dryer	on/off sensor	We will verify that the sensor powers on. Once we have verified the component works, we will verify that we can connect it to our smart hub by syncing the two devices through the appropriate firmware. Then we will install the sensor on the back of the dryer. Once we have the sensor installed, we will turn the dryer on to verify that a unique value is sent to the smart hub for off and on.
Rooms	Temperature sensor	We will verify that the sensor powers on. Once we have verified the component works, we will verify that we can connect it to our smart hub by syncing the two devices through the appropriate firmware. Then we will install the sensor

		in the appropriate room. We will verify that the sensor sends the current temperature of the room to the smart hub by verifying the current temperature of the room with a hand thermometer.
Rooms	Outlet controls	We will verify that the controller powers on. Once we have verified the component works, we will verify that we can connect it to our smart hub by syncing the two devices through the appropriate firmware. We will then install the control into its respective outlet. We will verify the outlet powers on and off and has a unique value for on and off by turning the outlet on and off via the smart hub. This will also verify that the smart hub can control the controller.
Rooms	Power usage sensor	We will verify that the sensor powers on. Once we have verified the component works, we will verify that we can connect it to our smart hub by syncing the two devices through the appropriate firmware. Then we will install the sensor in the appropriate room on the respected outlet or appliance. After that we will turn on devices plugged into the outlet in order to verify that an upwards counting value is being transmitted to the smart

		hub. This will verify that the sensor monitors the power consumption of the outlet.
A/C	Smart thermostat	We will verify that the thermostat powers on. Once we have verified that the component works, we will verify that we can connect it to our smart hub by syncing the two devices through the appropriate firmware. Then we will install the smart thermostat in place of the thermostat. After that we will verify that the smart thermostat sends the current temperature and temperature setting to the smart hub. Then we will verify that the smart hub can set the temperature setting of the smart thermostat.

Kitchen Appliances	on/off control	We will verify that the controller powers on. Once we have verified the component works, we will verify that we can connect it to our smart hub by syncing the two devices through the appropriate firmware. We will then install the control onto its respective appliance. We will verify the appliance powers on and off and has a unique value for on and off by turning the appliance on and off via the smart hub. This will also verify that the smart hub can control the controller.
Smart Hub	Raspberry Pi	We will verify that the Raspberry Pi powers on. Once we have verified the component works, we will verify that it can accept all of the appropriate firmware. After it has all of the firmware loaded and working, we will make test each of the hardware components against it to verify that they can talk to the smart hub and be controlled by it.
Android Application	Android Application	We will verify that the Android application can connect to the smart hub by checking the status of the hardware with the Android application and verifying that it is receiving accurate results by double checking against the actual status of the room, the status listed on the smart hub,

		and the status listed on the MySQL database. This will allow us to verify that the Android application receives its updates in real time.
Web Server	Apache Web Server	We will verify that the Apache web server appropriately relays information between the smart hub and the Android application via the MySQL database by checking the status of the hardware with the Android application and verifying that it is receiving accurate results by double checking against the actual status of the room, the status listed on the smart hub, and the status listed on the MySQL database. This will allow us to verify that the Apache web server processes its updates in real time.
Complete System	Doors open/close sensor	We will verify all of the door sensors work by checking the status of each individual sensor with the Android application and verifying that it is receiving accurate results by double checking against the actual status of the door, the status listed on the smart hub, and the status listed on the MySQL database. Opening and closing the door will allow us to verify the statuses are being updated in real time.
Complete System	Windows open/close sensor	We will verify all of the window sensors work by checking the status of each individual sensor with the Android application and verifying that it is receiving accurate results by double checking against the actual status of the window, the status listed on the smart hub, and the status listed on the MySQL database. Opening and closing the window will allow us to verify the statuses are being updated in real time.
Complete	Locks locked/	We will verify all of the locked/unlocked

System	unlocked control	controls work by checking the status of each individual lock with the Android application and verifying that it is receiving accurate results by double checking against the actual status of the lock, the status listed on the smart hub, and the status listed on the MySQL database. Locking and unlocking the lock will allow us to verify the statuses are being updated in real time.
Complete System	Lights on/off control	We will verify all of the light controls work by checking the status of each individual light with the Android application and verifying that it is receiving accurate results by double checking against the actual status of the light, the status listed on the smart hub, and the status listed on the MySQL database. Turning the light on and off will allow us to verify the statuses are being updated in real time.
Complete System	Fans on/off control	We will verify all of the fan controls work by checking the status of each individual fan with the Android application and verifying that it is receiving accurate results by double checking against the actual status of the fan, the status listed on the smart hub, and the status listed on the MySQL database. Turning the fan on and off will allow us to verify the statuses are being updated in real time.
Complete System	Doorways laser sensors	We will verify all of the doorway laser sensors work by checking the running counter for each room on the smart hub with the actual number of people found in each room. Changing the number of people in each room and double checking it against the running counter found on the smart hub, MySQL database, and Android application will allow us to verify the statuses are being

		updated in real time.
Complete System	Washer on/off sensor	We will verify that the washer sensor works by checking the status of the sensor with the Android application and verifying that it is receiving accurate results by double checking against the actual status of the washer, the status listed on the smart hub, and the status listed on the MySQL database. Turning the washer on and off will allow us to verify the statuses are being updated in real time.
Complete System	Dryer on/off sensor	We will verify that the dryer sensor works by checking the status of the sensor with the Android application and verifying that it is receiving accurate results by double checking against the actual status of the dryer, the status listed on the smart hub, and the status listed on the MySQL database. Turning the dryer on and off will allow us to verify the statuses are being updated in real time.
Complete System	Rooms temperature sensor	We will verify all of the temperature sensors work by checking the status of each individual sensor with the Android application and verifying that it is receiving accurate results by double checking against the actual temperature

		of the room, the status listed on the smart hub, and the status listed on the MySQL database. Monitoring the results over a period of time will verify that the temperatures are being updated in real time.
Complete System	Rooms outlet controls	We will verify all of the outlet controls work by checking the status of each individual outlet with the Android application and verifying that it is receiving accurate results by double checking against the actual status of the outlet, the status listed on the smart hub, and the status listed on the MySQL database. Turning the outlet on and off will allow us to verify the statuses are being updated in real time.
Complete System	Rooms power usage sensor	We will verify all of the power usage sensors work by checking the total usage for the month listed against the months power bill. Making sure that the total usage changes at the same rate on the smart hub, Android application, and MySQL database while devices are on will verify that the total usage is updated in real time.
Complete System	A/C smart thermostat	We will verify that the smart hub changes the temperature based on the results of the current average experienced

		temperature algorithms by checking the current temperature listed vs the temperature setting based on certain situations (i.e. no one home, everyone home, everyone in one area etc etc). We will also verify that the smart thermostat's current temperature setting can be changed by the Android application and is updated in real time on the MySQL database and smart hub.
Complete System	Kitchen appliances on/off controls	We will verify all of the appliance controls work by checking the status of each individual appliance with the Android application and verifying that it is receiving accurate results by double checking against the actual status of the appliance, the status listed on the smart hub, and the status listed on the MySQL database. Turning the appliance on and off will allow us to verify the statuses are being updated in real time.
Complete System	Smart Hub – Raspberry Pi	We will verify that all of the home automation functions of the system work. This entails us verifying all of the above tests and also: verifying that the locks lock when no one is home, that the air conditioning adjusts based on the presence of users, that the rooms turn on when someone enters them for the first time, that the rooms shut off when no one has been in them for 20 minutes, that the system unlocks the locks when the first user connects to wifi, and that the status of all of the hardware components are actively monitored when no one is home.
Complete System	Android Application	We will verify that the Android application can check the status of and control all of the hardware components of the system.

Complete System	Web Server	We will verify that the Android application can check the status of and control all of the hardware components of the system. We will also verify that the statuses are updated on the MySQL database appropriately along with the statuses on the smart hub.
Complete System	Complete System	We will verify that two Android applications can check the status of and control all of the hardware components of the system concurrently while checking the status updates on the smart hub and MySQL database. This will conclude the testing of our system.

8.0 Project Operation

The Close to Home Project was in full operation at the Senior Design Day fair located on the UCF Campus. What was configured was a sample Lock Module installed in a small door, a 120V module, and an in-wall module. The router on the left, combined with the laptop on the right combine to function as a sort of “pseudo-internet” for the online database to be hosted on. Instead of relying on a saturated wireless network, where we do not have control over the port forwarding or permissions, our group decided to instead create a local intranet without true internet access, and simply operate the cell phones and entire Close to Home system within in. This proved to be very reliable, and we were able to get tons of operation out of the system, throughout the day as we displayed the operation off to many passers by. You can see the full configuration below:

Figure 8.0.1 : The Close to Home prototype system on display

Below, we will provide the user with step-by-step instructions to operate the prototype of the Close to Home system. For this to work, you will need the Hub, a router, a laptop with the database installed, an android phone with the Close to Home android app installed, and the 3 modules to be powered by batteries. Once you've acquired all of these, follow the below instructions:

1. Provide Power to the Router, and connect the laptop to its Wi-Fi.
2. Launch the database from the laptop once it is connected.
3. Once the database is online, provide power to the Hub.
4. Give the Hub approx 2 minutes to come online.
5. Launch the Close to Home Android App from the smart phone.
6. You should now be able to easily operate the locks and lights.

For a demonstration video of the Close to Home system in action, feel free to search Youtube for a video titled “Close 2 Home Project Demo”, which is a short demo that we created to show two of our committee members when they could not attend.

9.0 Project and Group Organization

We will now take the time to go into further detail as to how our group composition was arranged, and how we met up throughout the two semesters in order to make the Close to Home System a reality. This will be a rather broad overview, with some finer details here and there. What follows is a list of the four members of the Close to Home team, their intrinsic skills, and how they helped contribute to the final product of this Senior Design course:

- Nicholas Godfrey – Major: Electrical Engineering – EE
- Joshua Early – Major: Computer Engineering – CpE
- Daniel Krummen – Major: Computer Engineering – CpE
- Marc Garcia – Major: Computer Engineering – CpE

These four members each possessed unique skills and came from a background that provided a special spin on the aspect of the Close to Home system that they had been assigned to work on. Though each member was working on their individual aspects, all parts of the Close to Home system had to function in harmony, so group meet-ups and constant coordination were necessary to get the ball rolling on various aspects of this project. Through their combined efforts and individual skills, it was possible to realize the result of the Close to Home project, as a well-crafted home monitoring and security system.

Nicholas Godfrey is currently pursuing a bachelors in Electrical Engineering at the University of Central Florida. He is a devoted member of the Theta Tau fraternity, and is highly skilled with electrical circuits. Given his unique expertise with circuits, at least among a group of Computer Engineers, he chose to take on the role of developing the in-house wall modules. These modules would be used to monitor the various entryways, windows, devices, and appliances in your everyday home. He designed simple circuits that would receive power from the house's grid, convert it accordingly, and then use them on a small board equipped with an MSP430 Microprocessor. From there, the data pertaining to the observed module would be transmitted via ZigBee to the central Hub of communications for the Close to Home system. Since Nicholas had to develop the circuit from scratch, his expertise in step-down circuits, and breadboard prototyping was a brilliant asset to the Senior Design team, and we could not have done it without him.

Joshua Early is currently pursuing a bachelors in Computer Engineering at the University of Central Florida. He is a hard worker with an affinity for mobile and object-oriented software development, so he decided to devote his work-hours, alongside Marc Garcia, to the development of the Android Application that would be the control system for the end-user in the Close to Home system. Getting the Android Application to interface directly with the Close to Home Hub, and then to interface with a Server Application of Marc's design, was no easy task. Ensuring that all devices were speaking the same language, and working in unison was something that took hours of labored testing as well as heavy trial and error. However, the final product will yield an Android Application that will be convenient to use, and will provide full control and vision over your home while you're in it, or if you're away.

Daniel Krummen is pursuing a bachelors in Computer Engineering at the University of Central Florida. Daniel expressed an interest early on into the field of microcontrollers and lower-level programming, so he decided to work on the central Hub of the Close to Home system. The hub is a developmental board known as the Raspberry Pi, that provides GPIO headers for the Zigbee communication that we will be using, and a robust processor and hefty amount of RAM. These system resources will prove invaluable when computing the various kinds of calculations necessary to schedule the Close to Home mainframe. The Close to Home system will feature a fully customizable method of automating your house's appliances to ensure that there are rarely lights on in an unoccupied room, or a television running with no one watching. Daniel plans to incorporate either built-in profiles for recommended house monitoring settings, or providing the client with a way to customize their own configuration via the Android Application.

Marc Garcia is pursuing a bachelors in Computer Engineering at the University of Central Florida. Given Marc's expertise with Object-Oriented and higher level programming, he and Josh worked together to create the most aesthetically appealing Android Application (alliteration, eh?) for the end user, as well as making sure that, along the way, the application always had the goal of being user-friendly in mind. Marc had a great eye for design, and was the specialist when any of the project needed a professional polish on it. For instance, the GUI of the Android Application was the product of Marc's design, and making sure that it ran smoothly on nearly all Android platforms was another of Marc's goals.

The group composition that we had for this project could not have been more ideal. The team had a great range of experience in various fields, and the varied kinds of expertise lent itself well to a polished and effective finished product. If anything, it might have been beneficial to have another Electrical Engineering major in our group to assist with Nicholas' tasks, but he was able to shoulder the burden of a majority of the electrical work, with occasional assistance from the other members. The end result was a project that was effective at providing its end user with a home monitoring and security system to help them save on energy costs.

Throughout the semesters, it was important to establish weekly meeting times, perhaps more than one per week on occasion, in order to put in the extra man hours to make the Close to Home system a reality. Meetings were generally held within the Atrium of the Engineering 2 building, but occasionally took place in the meeting rooms of Engineering 1 Room 247—the computer lab that requires card access. Having a white board was beneficial to sketch out prototype schematics and to kind of storyboard the entire diagram of the Close to Home system as a whole.

Meetings also spontaneously took place after class on days where the professor revealed information that required additional hours devoted to the project. Most often, the hours following class allowed for several of the group members to be available and to work on the finer details of the project. Without putting in the extra effort, coming to a head and creating the finished product would not have been possible, and it shows with our additional man hours.

Each member of the Close to Home team contributed with their unique set of skills and impressive level of dedication to making this project a reality. What follows below is a general summary of the amount of time each member of the Close to Home team spent working on the project:

	Main Module Worked On	Hours Spent
Nicholas	C2H Modules	72
Josh	Android Application	66
Daniel	C2H Hub	69
Marc	Android Application	66

Table 9.1: Group Contribution by member

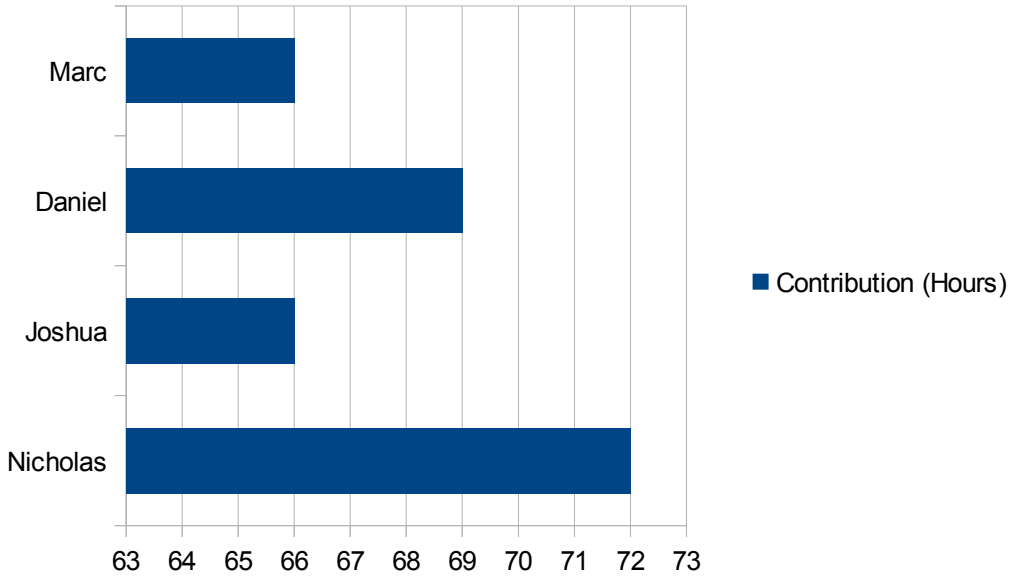


Figure 9.2 Chart Representation of the Above Table

When members were unable to meet in person, or after class, we had a Google Group formed with all members in attendance. A majority of the consultation and coordination occurred in this setting, and the team was able to communicate well given the convenient medium of the Google Group. Given that the project was so modular and required effort from diverse parties all coordinating together, the Close to Home system would not have been possibly created without a close-knit way to talking to one another.

Whenever we met via the Google Groups interface, we could truly feel like we were getting a lot of work done. Through the web communication and having access to our own home devices was ideal for what we were looking to get done in the amount of time that we had. You can see our productivity here in the middle graph displayed below:

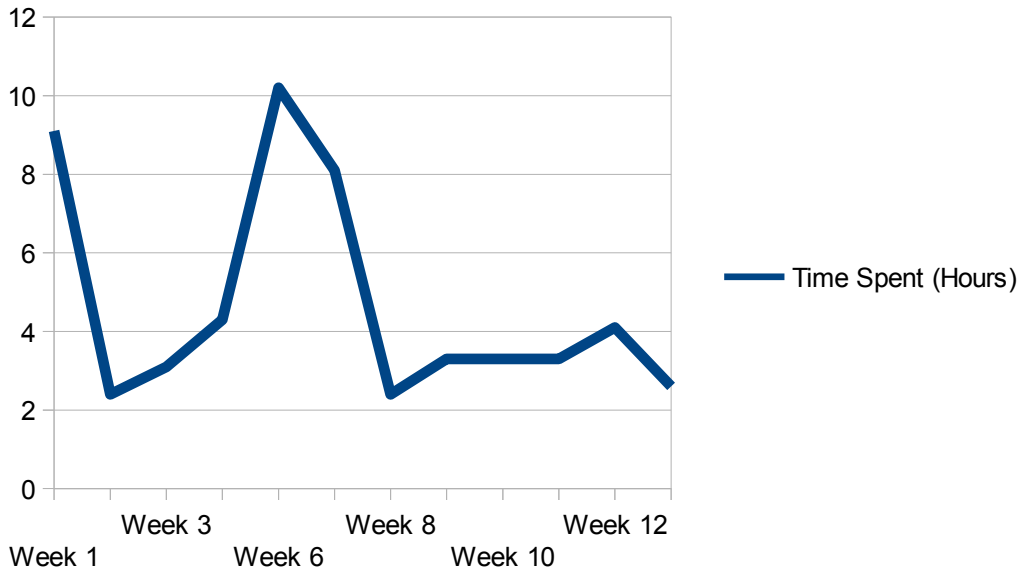


Figure 9.3: Hours Logged on Project via Google Groups Communication

Without the advent of Google Groups, it would have been difficult to near impossible to make the Close to Home project a reality, and contributing equally to the group effort was every team member of the Senior Design Close to Home team. The equal contribution of every team member was essentially vital to the group effort and success of the group as a whole and for the entire two semesters the amount of hard work, blood, sweat and man-hours that went into this project is truly something to be proud of. I cannot wait for the hands on work in the second semester, as it seems like it will be something quite interesting to do instead of getting the documentation out of the way. Hands on experience will be a great asset for us in the long run.

10.0 Project Summary and Conclusions

The Close to Home Project's ultimate goal is provide the end-user with a product that they can rely on. We want to have a situation wherein the user is able to safely leave the house with the peace of mind that no appliances are running when they're not supposed to, the doors and windows are properly locked, and ensuring that they are getting the maximum in power savings per month without going through the hassle of manually checking. A smart home should be just that: smart. Our Close to Home system is that and so much more. We made the focus of our project on the energy savings that our system could provide, as well as a strong emphasis on making it as user-friendly as possible. Of course, it was also necessary for the system to be quick and efficient in its operation, which are two qualities that the Close to Home system excels at.

The Close to Home system set out to provide its users with a reliable, efficient system that can be integrated into their daily lives. With Close to Home installed in a household, many worries and stress can be eliminated, and provide the user with a more relaxing home experience than ever before. Given that our Close to Home system yields the power gains that we first provided in our Requirements section, and the Android Application will be robust and easy to use, the Close to Home system has accomplished all that it set out to do.

In summary, this project was a wonderful experience for all team members involved. We quickly learned the importance of every week as things got closer and closer to our deadlines. It truly showed that you get out what you put in, and the Close to Home project is 100% composed of our group efforts and determination. We hope that in the future we can list our Senior Design project on all of our individual résumés. It will hopefully help us springboard our career into the field of either integrated circuits, software engineering, or microprocessor engineering. We are very thankful for our professor, Dr. Richie for giving us this opportunity to show our stuff in a group setting in order to create a solid foundation for our future. We also would like to thank Duke Energy for providing us with the funding that we needed to make our dream of the C2H system a reality.